

# **PHP Manual**

**Stig Sæther Bakken**

**Alexander Aulbach**

**Egon Schmid**

**Jim Winstead**

**Lars Torben Wilson**

**Rasmus Lerdorf**

**Zeev Suraski**

Edited by  
**Stig Sæther Bakken**

**PHP Manual**

by Stig Sæther Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, and Zeev Suraski

Edited by Stig Sæther Bakken

Published 1999-11-03

Copyright © 1997, 1998, 1999 by the PHP Documentation Group

**Copyright**

This manual is © Copyright 1997, 1998, 1999 the PHP Documentation Group. The members of this group are listed on the front page of this manual.

This manual can be redistributed under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

# Dedication

For Christine and Rasmus

1999-11-03

# Table of Contents

<b>Preface .....</b>	<b>29</b>
About this Manual.....	29
<b>I. Getting Started .....</b>	<b>30</b>
1. Introduction .....	31
What is PHP?.....	31
What can PHP do?.....	31
A brief history of PHP .....	32
2. Installation.....	33
Downloading the latest version .....	33
Installation on UNIX systems .....	33
Quick Installation Instructions (Apache Module Version) .....	33
Configuration .....	34
Apache module .....	34
fhttpd module.....	35
CGI version .....	35
Database Support Options.....	35
Adabas D .....	35
dBase .....	35
filePro .....	36
mSQL .....	36
MySQL.....	36
iODBC.....	36
OpenLink ODBC.....	37
Oracle .....	37
PostgreSQL .....	37
Solid .....	37
Sybase.....	38
Sybase-CT .....	38
Velocis .....	38
A custom ODBC library.....	38
Unified ODBC.....	39
LDAP.....	39
Other configure options.....	40
--with-mcrypt= <i>DIR</i> .....	40
--enable-sysvsem.....	40
--enable-sysvshm .....	40
--with-xml .....	40
--enable-maintainer-mode .....	40

-with-system-regex .....	40
-with-config-file-path.....	41
-with-exec-dir .....	41
-enable-debug .....	41
-enable-safe-mode .....	41
-enable-track-vars .....	42
-enable-magic-quotes.....	42
-enable-debugger .....	42
-enable-discard-path .....	42
-enable-bcmath .....	43
-enable-force-cgi-redirect .....	43
-disable-short-tags .....	43
-enable-url-includes .....	43
-disable-syntax-hl .....	44
CPPFLAGS and LDFLAGS.....	44
Building .....	44
Testing.....	44
Benchmarking .....	44
Installation on Windows 95/98/NT systems.....	45
General Installation Steps .....	45
Windows 95/98/NT and PWS/IIS 3.....	46
Windows NT and IIS 4 .....	47
Windows 9x/NT and Apache 1.3.x .....	47
Omni HTTPd 2.0b1 for Windows.....	48
PHP Modules .....	48
Problems? .....	49
Read the FAQ.....	49
Bug reports.....	49
Other problems.....	49
3. Configuration .....	51
The configuration file .....	51
General Configuration Directives .....	51
Mail Configuration Directives .....	56
Safe Mode Configuration Directives.....	56
Debugger Configuration Directives .....	57
Extension Loading Directives .....	57
MySQL Configuration Directives .....	58
mSQL Configuration Directives .....	58
Postgres Configuration Directives .....	58
Sybase Configuration Directives.....	59

Sybase-CT Configuration Directives .....	59
Informix Configuration Directives .....	60
BC Math Configuration Directives .....	61
Browser Capability Configuration Directives .....	62
Unified ODBC Configuration Directives .....	62
4. Security .....	63
CGI binary .....	63
Possible attacks .....	63
Case 1: only public files served .....	64
Case 2: using –enable-force-cgi-redirect .....	64
Case 3: setting doc_root or user_dir .....	64
Case 4: PHP parser outside of web tree .....	65
Apache module .....	66
<b>II. Language Reference .....</b>	<b>67</b>
5. Basic syntax .....	68
Escaping from HTML .....	68
Instruction separation .....	68
Comments .....	69
6. Types .....	70
Integers .....	70
Floating point numbers .....	70
Strings .....	70
String conversion .....	71
Arrays .....	72
Single Dimension Arrays .....	72
Multi-Dimensional Arrays .....	73
Objects .....	74
Object Initialization .....	74
Type juggling .....	75
Type casting .....	75
7. Variables .....	77
Variable scope .....	77
Variable variables .....	79
Variables from outside PHP .....	80
HTML Forms (GET and POST) .....	80
IMAGE SUBMIT variable names .....	81
HTTP Cookies .....	81
Environment variables .....	82
Determining variable types .....	82
8. Constants .....	83

9. Expressions .....	85
10. Operators .....	88
Arithmetic Operators .....	88
String Operators .....	88
Assignment Operators .....	88
Bitwise Operators .....	89
Logical Operators .....	89
Comparison Operators .....	90
Operator Precedence .....	91
11. Control Structures .....	93
if .....	93
else .....	93
elseif .....	94
Alternative syntax for if structures: <code>if(): ... endif;</code> .....	95
while .....	95
do..while .....	96
for .....	97
break .....	99
continue .....	99
switch .....	100
require .....	102
include .....	102
12. Functions .....	104
User-defined functions .....	104
Returning values .....	104
Function arguments .....	104
Making arguments be passed by reference .....	105
Default argument values .....	106
old_function .....	107
13. Classes and Objects .....	108
class .....	108
<b>III. Features .....</b>	<b>111</b>
14. Error handling .....	112
15. Creating GIF images .....	113
16. HTTP authentication with PHP .....	114
17. Cookies .....	116
18. Handling file uploads .....	117
POST method uploads .....	117
Common Pitfalls .....	118
Uploading multiple files .....	118

PUT method support.....	119
19. Using remote files .....	121
20. Connection handling .....	123
21. Persistent database connections .....	125
<b>IV. Function Reference .....</b>	<b>127</b>
I. Adabas D functions.....	128
ada_afetch.....	129
ada_autocommit .....	129
ada_close .....	129
ada_commit .....	129
ada_connect.....	130
ada_exec .....	130
ada_fetchrow .....	130
ada_fieldname.....	131
ada_fieldnum .....	131
ada_fieldtype.....	131
ada_freeresult .....	132
ada_numfields.....	132
ada_numrows.....	132
ada_result.....	133
ada_resultall.....	133
ada_rollback .....	133
II. Apache-specific functions .....	135
apache_lookup_uri .....	136
apache_note .....	137
getallheaders .....	137
virtual.....	138
III. Array functions .....	139
array.....	140
array_count_values.....	140
array_keys.....	141
array_merge.....	142
array_pad .....	143
array_pop .....	143
array_push .....	144
array_reverse .....	145
array_shift.....	146
array_slice.....	147
array_splice.....	147
array_unshift.....	149

array_values.....	150
array_walk .....	150
arsort .....	152
asort .....	152
compact.....	153
count .....	154
current.....	155
each.....	156
end .....	157
extract .....	158
in_array.....	159
key .....	160
krsort.....	161
ksort .....	161
list .....	162
next .....	163
pos.....	164
prev .....	164
range .....	165
reset.....	165
rsort.....	166
shuffle .....	167
sizeof.....	167
sort .....	168
uasort .....	169
uksort .....	169
usort .....	170
IV. Aspell functions .....	172
aspell_new .....	173
aspell_check .....	173
aspell_check-raw .....	174
aspell_suggest.....	174
V. Arbitrary precision mathematics functions .....	176
bcadd.....	177
bccomp .....	177
bcddiv .....	177
bcmmod .....	178
bcmul .....	178
bcpow.....	179
bcscale .....	179

bcsqrt .....	180
bcsub .....	180
VI. Calendar functions .....	182
JDToGregorian .....	183
GregorianToJD .....	183
JDToJulian .....	184
JulianToJD .....	184
JDToJewish .....	185
JewishToJD .....	185
JDToFrench .....	186
FrenchToJD .....	186
JDMonthName .....	187
JDDayOfWeek .....	187
easter_date .....	188
easter_days .....	189
VII. ClibPDF functions .....	191
cpdf_set_creator .....	195
cpdf_set_title .....	195
cpdf_set_subject .....	195
cpdf_set_keywords .....	196
cpdf_open .....	196
cpdf_close .....	197
cpdf_page_init .....	198
cpdf_finalize_page .....	198
cpdf_finalize .....	199
cpdf_output_buffer .....	199
cpdf_save_to_file .....	200
cpdf_set_current_page .....	200
cpdf_begin_text .....	201
cpdf_end_text .....	202
cpdf_show .....	202
cpdf_show_xy .....	203
cpdf_text .....	203
cpdf_set_font .....	204
cpdf_set_leading .....	204
cpdf_set_text_rendering .....	205
cpdf_set_horiz_scaling .....	205
cpdf_set_text_rise .....	206
cpdf_set_text_matrix .....	206
cpdf_set_text_pos .....	207

cpdf_set_char_spacing .....	207
cpdf_set_word_spacing .....	208
cpdf_continue_text .....	208
cpdf_stringwidth.....	209
cpdf_save .....	209
cpdf_restore .....	210
cpdf_translate .....	211
cpdf_scale .....	211
cpdf_rotate.....	211
cpdf_setflat .....	212
cpdf_setlinejoin .....	212
cpdf_setlinecap.....	213
cpdf_setmiterlimit.....	213
cpdf_setlinewidth.....	214
cpdf_setdash .....	214
cpdf_moveto .....	215
cpdf_rmoveto.....	215
cpdf_curveto .....	216
cpdf_lineto.....	216
cpdf_rlineto.....	217
cpdf_circle .....	217
cpdf_arc .....	218
cpdf_rect.....	219
cpdf_closepath.....	219
cpdf_stroke .....	219
cpdf_closepath_stroke .....	220
cpdf_fill.....	220
cpdf_fill_stroke.....	221
cpdf_closepath_fill_stroke.....	221
cpdf_clip .....	222
cpdf_setgray_fill.....	222
cpdf_setgray_stroke.....	223
cpdf_setgray .....	223
cpdf_setrgbcolor_fill.....	224
cpdf_setrgbcolor_stroke .....	224
cpdf_setrgbcolor .....	225
cpdf_add_outline .....	225
cpdf_set_page_animation .....	226
cpdf_import_jpeg .....	227
cpdf_place_inline_image.....	228

cpdf_add_annotation .....	228
VIII. Date and Time functions .....	230
checkdate .....	231
date .....	231
getdate.....	233
gettimeofday .....	234
gmdate .....	234
gmmktime.....	235
gmstrftime.....	236
microtime.....	236
mktime .....	237
strftime.....	238
time .....	239
IX. Database (dbm-style) abstraction layer functions .....	241
dba_close .....	243
dba_delete.....	243
dba_exists .....	244
dba_fetch .....	244
dba_firstkey .....	245
dba_insert .....	245
dba_nextkey.....	246
dba_popen.....	246
dba_open.....	247
dba_optimize .....	248
dba_replace.....	248
dba_sync .....	249
X. dBase functions .....	251
dbase_create .....	252
dbase_open .....	253
dbase_close.....	253
dbase_pack .....	254
dbase_add_record .....	254
dbase_replace_record .....	255
dbase_delete_record .....	255
dbase_get_record.....	256
dbase_get_record_with_names.....	256
dbase_numfields .....	257
dbase_numrecords .....	258
XI. dbm functions .....	259
dbmopen .....	260

dbmclose.....	260
dbmexists.....	261
dbmfetch.....	261
dbminsert.....	261
dbmreplace.....	262
dbmdelete.....	262
dbmfirstkey.....	263
dbmnextkey.....	263
dblist.....	264
XII. Directory functions.....	265
chdir.....	266
dir.....	266
closedir.....	267
opendir.....	267
readdir.....	267
rewinddir.....	268
XIII. Dynamic Loading functions.....	270
dl.....	271
XIV. Program Execution functions.....	272
escapeshellcmd.....	273
exec.....	273
system.....	274
passthru.....	274
XV. Forms Data Format functions.....	276
fdf_open.....	278
fdf_close.....	278
fdf_create.....	279
fdf_save.....	280
fdf_get_value.....	280
fdf_set_value.....	281
fdf_next_field_name.....	281
fdf_set_ap.....	282
fdf_set_status.....	282
fdf_get_status.....	283
fdf_set_file.....	283
fdf_get_file.....	284
XVI. filePro functions.....	285
filepro.....	286
filepro_fieldname.....	286
filepro_fieldtype.....	286

filepro_fieldwidth .....	287
filepro_retrieve.....	287
filepro_fieldcount.....	288
filepro_rowcount.....	288
XVII. Filesystem functions .....	290
basename .....	291
chgrp .....	291
chmod .....	292
chown.....	292
clearstatcache.....	293
copy .....	294
delete.....	294
dirname .....	295
diskfreespace .....	296
fclose.....	296
feof.....	297
fgetc .....	297
fgetcsv.....	298
fgets .....	299
fgetss.....	300
file .....	300
file_exists .....	301
fileatime .....	301
filectime .....	302
filegroup.....	302
fileinode .....	303
filemtime.....	303
fileowner .....	304
fileperms .....	304
filesize.....	305
filetype .....	305
flock .....	306
fopen.....	307
fpassthru .....	308
fputs .....	309
fread.....	309
fseek.....	310
ftell.....	311
fwrite.....	311
set_file_buffer .....	312

is_dir .....	312
is_executable .....	313
is_file .....	313
is_link .....	314
is_readable .....	314
is_writable .....	315
link .....	316
linkinfo .....	316
mkdir .....	317
pclose .....	317
popen .....	318
readfile .....	318
readlink .....	319
rename .....	320
rewind .....	320
rmdir .....	321
stat .....	321
lstat .....	322
symlink .....	323
tempnam .....	324
touch .....	324
umask .....	325
unlink .....	325
XVIII. FTP functions .....	327
ftp_connect .....	328
ftp_login .....	328
ftp_pwd .....	328
ftp_cdup .....	329
ftp_chdir .....	329
ftp_mkdir .....	330
ftp_rmdir .....	330
ftp_nlist .....	331
ftp_rawlist .....	331
ftp_systype .....	332
ftp_pasv .....	332
ftp_get .....	333
ftp_fget .....	333
ftp_put .....	334
ftp_fput .....	334
ftp_size .....	335

ftp_mdtm .....	335
ftp_rename .....	336
ftp_delete .....	336
ftp_quit .....	337
XIX. HTTP functions .....	338
header .....	339
setcookie .....	339
XX. Hyperwave functions .....	342
hw_Array2Objrec .....	347
hw_Children .....	347
hw_ChildrenObj .....	347
hw_Close .....	348
hw_Connect .....	348
hw_Cp .....	349
hw_Deleteobject .....	350
hw_DocByAnchor .....	350
hw_DocByAnchorObj .....	350
hw_DocumentAttributes .....	351
hw_DocumentBodyTag .....	351
hw_DocumentContent .....	352
hw_DocumentSetContent .....	352
hw_DocumentSize .....	353
hw_ErrorMsg .....	353
hw_EditText .....	354
hw_Error .....	354
hw_Free_Document .....	355
hw_GetParents .....	355
hw_GetParentsObj .....	356
hw_GetChildColl .....	356
hw_GetChildCollObj .....	357
hw_GetRemote .....	357
hw_GetRemoteChildren .....	358
hw_GetSrcByDestObj .....	359
hw_GetObject .....	359
hw_GetAndLock .....	360
hw_GetText .....	361
hw_GetObjectByQuery .....	362
hw_GetObjectByQueryObj .....	362
hw_GetObjectByQueryColl .....	363
hw_GetObjectByQueryCollObj .....	363

hw_GetChildDocColl .....	364
hw_GetChildDocCollObj .....	364
hw_GetAnchors .....	365
hw_GetAnchorsObj .....	365
hw_Mv .....	366
hw_Identify .....	366
hw_InCollections .....	367
hw_Info .....	368
hw_InsColl .....	368
hw_InsDoc .....	368
hw_InsertDocument .....	369
hw_InsertObject .....	369
hw_mapid .....	370
hw_Modifyobject .....	371
hw_New_Document .....	373
hw_Objrec2Array .....	374
hw_OutputDocument .....	375
hw_pConnect .....	375
hw_PipeDocument .....	376
hw_Root .....	376
hw_Unlock .....	377
hw_Who .....	377
hw_Username .....	377
XXI. Image functions .....	379
GetImageSize .....	380
ImageArc .....	381
ImageChar .....	381
ImageCharUp .....	382
ImageColorAllocate .....	382
ImageColorAt .....	383
ImageColorClosest .....	383
ImageColorExact .....	384
ImageColorResolve .....	384
ImageColorSet .....	385
ImageColorsForIndex .....	385
ImageColorsTotal .....	386
ImageColorTransparent .....	386
ImageCopyResized .....	387
ImageCreate .....	387
ImageCreateFromGif .....	387

ImageDashedLine .....	388
ImageDestroy .....	389
ImageFill.....	389
ImageFilledPolygon .....	390
ImageFilledRectangle.....	390
ImageFillToBorder .....	391
ImageFontHeight .....	391
ImageFontWidth .....	392
ImageGif.....	392
ImageInterlace .....	393
ImageLine.....	394
ImageLoadFont.....	394
ImagePolygon.....	395
ImagePSBBox .....	395
ImagePSEncodeFont .....	396
ImagePSFreeFont .....	397
ImagePSLoadFont .....	397
ImagePSText.....	398
ImageRectangle .....	399
ImageSetPixel.....	400
ImageString .....	400
ImageStringUp .....	401
ImageSX .....	401
ImageSY .....	402
ImageTTFBBox.....	402
ImageTTFTText.....	403
XXII. IMAP functions .....	406
imap_append.....	407
imap_base64 .....	407
imap_body .....	407
imap_check.....	408
imap_close.....	409
imap_createmailbox .....	409
imap_delete.....	410
imap_deletemailbox .....	410
imap_expunge.....	410
imap_fetchbody .....	411
imap_fetchstructure .....	412
imap_header .....	412
imap_headers .....	415

imap_listmailbox .....	415
imap_getmailboxes .....	416
imap_listsubscribed .....	416
imap_getsubscribed .....	417
imap_mail_copy .....	417
imap_mail_move .....	418
imap_num_msg .....	419
imap_num_recent .....	419
imap_open .....	419
imap_ping .....	420
imap_renamemailbox .....	421
imap_reopen .....	421
imap_subscribe .....	422
imap_undelete .....	423
imap_unsubscribe .....	423
imap_qprint .....	423
imap_8bit .....	424
imap_binary .....	424
imap_scanmailbox .....	425
imap_mailboxmsginfo .....	425
imap_rfc822_write_address .....	426
imap_rfc822_parse_adrlist .....	427
imap_setflag_full .....	427
imap_clearflag_full .....	428
imap_sort .....	429
imap_fetchheader .....	430
imap_uid .....	430
imap_msgno .....	431
imap_search .....	431
imap_last_error .....	433
imap_errors .....	433
imap_alerts .....	434
imap_status .....	434
XXIII. PHP options & information .....	436
error_log .....	437
error_reporting .....	438
extension_loaded .....	439
getenv .....	440
get_cfg_var .....	440
get_current_user .....	441

get_magic_quotes_gpc .....	441
get_magic_quotes_runtime.....	442
getlastmod.....	442
getmyinode .....	443
getmypid .....	444
getmyuid .....	444
getrusage.....	444
phpinfo.....	445
phpversion .....	446
putenv .....	446
set_magic_quotes_runtime .....	447
set_time_limit .....	447
XXIV. Informix functions.....	449
ifx_connect .....	451
ifx_pconnect .....	451
ifx_close .....	452
ifx_query.....	453
ifx_prepare.....	454
ifx_do.....	455
ifx_error .....	456
ifx_errormsg .....	457
ifx_affected_rows .....	457
ifx_getsqlca.....	458
ifx_fetch_row.....	459
ifx_htmltbl_result .....	461
ifx_fieldtypes .....	462
ifx_fieldproperties.....	462
ifx_num_fields.....	463
ifx_num_rows.....	464
ifx_free_result.....	464
ifx_create_char .....	465
ifx_free_char.....	465
ifx_update_char .....	465
ifx_get_char.....	466
ifx_create_blob.....	466
ifx_copy_blob.....	467
ifx_free_blob .....	467
ifx_get_blob.....	468
ifx_update_blob.....	468
ifx_blobinfile_mode .....	469

ifx_textasvarchar .....	469
ifx_byteasvarchar.....	470
ifx_nullformat.....	470
ifxus_create_slob.....	471
ifx_free_slob.....	471
ifxus_close_slob .....	472
ifxus_open_slob.....	472
ifxus_tell_slob .....	473
ifxus_seek_slob .....	473
ifxus_read_slob.....	474
ifxus_write_slob .....	474
XXV. InterBase functions.....	476
ibase_connect .....	477
ibase_pconnect .....	477
ibase_close.....	477
ibase_query.....	478
ibase_fetch_row.....	478
ibase_free_result.....	478
ibase_prepare .....	479
ibase_bind.....	479
ibase_execute.....	480
ibase_free_query.....	480
ibase_timefmt .....	480
XXVI. LDAP functions.....	482
ldap_add .....	485
ldap_mod_add .....	486
ldap_mod_del .....	486
ldap_mod_replace.....	487
ldap_bind .....	487
ldap_close .....	488
ldap_connect.....	488
ldap_count_entries.....	489
ldap_delete.....	489
ldap_dn2ufn.....	490
ldap_explode_dn.....	490
ldap_first_attribute.....	491
ldap_first_entry.....	491
ldap_free_result .....	492
ldap_get_attributes.....	492
ldap_get_dn .....	494

ldap_get_entries.....	494
ldap_get_values .....	495
ldap_get_values_len .....	496
ldap_list .....	497
ldap_modify.....	498
ldap_next_attribute .....	499
ldap_next_entry .....	499
ldap_read .....	500
ldap_search.....	500
ldap_unbind .....	502
ldap_err2str.....	502
ldap_errno.....	503
ldap_error .....	504
<b>V. Appendixes.....</b>	<b>506</b>
A. Migrating from PHP/FI 2.0 to PHP 3.0.....	507
About the incompatibilities in 3.0 .....	507
Start/end tags .....	507
if..endif syntax .....	508
while syntax.....	509
Expression types.....	509
Error messages have changed.....	510
Short-circuited boolean evaluation .....	510
Function true/false return values .....	510
Other incompatibilities .....	511
B. PHP development .....	513
Adding functions to PHP3.....	513
Function Prototype.....	513
Function Arguments.....	513
Variable Function Arguments .....	514
Using the Function Arguments .....	514
Memory Management in Functions .....	515
Setting Variables in the Symbol Table .....	516
Returning simple values.....	518
Returning complex values.....	519
Using the resource list.....	520
Using the persistent resource table .....	521
Adding runtime configuration directives .....	523
Calling User Functions .....	524
HashTable *function_table .....	524
pval *object.....	524

pval *function_name.....	524
pval *retval.....	524
int param_count .....	525
pval *params[] .....	525
Reporting Errors .....	525
E_NOTICE.....	525
E_WARNING .....	525
E_ERROR.....	525
E_PARSE.....	526
E_CORE_ERROR .....	526
E_CORE_WARNING.....	526
C. The PHP Debugger.....	527
Using the Debugger .....	527
Debugger Protocol.....	527

## List of Tables

2-1. PHP Modules.....	48
6-1. Escaped characters .....	71
10-1. Arithmetic Operators .....	88
10-2. Bitwise Operators .....	89
10-3. Logical Operators .....	90
10-4. Comparison Operators.....	90
10-5. Operator Precedence.....	91
1. Calendar modes .....	187
1. Calendar week modes .....	188
1. Font file format .....	395
1. error_log log types .....	437
1. error_reporting bit values.....	439
B-1. PHP Internal Types.....	514
C-2. Debugger Error Types.....	528

## List of Examples

1-1. An introductory example.....	31
5-1. Ways of escaping from HTML.....	68
7-1. Simple form variable .....	80
7-2. More complex form variables .....	80
7-3. SetCookie Example .....	81
8-1. Defining Constants .....	84
8-2. Using __FILE__ and __LINE__ .....	84
15-1. GIF creation with PHP .....	113
16-1. HTTP Authentication example.....	114
16-2. HTTP Authentication example forcing a new name/password .....	115
18-1. File Upload Form .....	117
18-2. Uploading multiple forms .....	118
19-1. Getting the title of a remote page .....	121
19-2. Storing data on a remote server .....	121
1. getallheaders() Example .....	138
1. array example .....	140
1. array_count_values example .....	141
1. array_keys example .....	141
1. array_merge example .....	142
1. array_pad example .....	143

1. array_pop example .....	144
1. array_push example .....	145
1. array_reverse example.....	??
1. array_shift example .....	146
1. array_slice examples .....	147
1. array_splice examples .....	148
1. array_unshift example.....	149
1. array_values example.....	150
1. array_walk example .....	151
1. arsort example.....	152
1. asort example .....	153
1. compact example.....	154
1. each examples.....	156
2. Traversing \$HTTP_POST_VARS with each.....	157
1. extract example .....	159
1. in_array example.....	160
1. krsort example.....	161
1. ksort example .....	162
1. list example .....	163
1. rsort example .....	166
1. shuffle example.....	167
1. sort example .....	168
1. uksort example.....	169
1. usort example .....	170
1. aspell_new .....	173
1. aspell_check.....	173
1. aspell_check_raw.....	174
1. aspell_suggest.....	175
1. Calendar functions .....	183
1. easter_date example .....	189
1. easter_date example .....	189
1. Text output .....	201
1. Text output .....	202
1. Save/Restore .....	210
1. Adding a page outline.....	226
1. date example .....	232
2. date and mktime example .....	233
1. gmdate example.....	235
1. gmstrftime example .....	236
1. mktime example.....	237

1. strftime example.....	239
1. Creating a dBase database file .....	252
1. Using dbase_numfields .....	257
1. Visiting every key/value pair in a dbm database.....	264
1. Dir() Example .....	266
1. List all files in the current directory.....	268
1. Accessing the form data.....	278
1. Populating a PDF document .....	279
1. basename example.....	291
1. copy example .....	294
1. dirname example.....	295
1. diskfreespace example.....	296
1. fgetcsv() example - Read and print entire contents of a CSV file .....	298
1. Reading a file line by line .....	299
1. fopen() example .....	308
1. tempnam() example .....	324
1. setcookie examples.....	340
1. modifying an attribute.....	372
2. adding a completely new attribute .....	372
3. modifying Title attribute .....	372
4. modifying Title attribute .....	372
5. removing attribute .....	373
1. GetImageSize.....	380
2. GetImageSize returning IPTC .....	380
1. Example to handle an error during creation (courtesy vic@zymsys.com ).....	388
1. ImageTTFTText .....	404
1. error_log examples.....	438
1. getlastmod() example.....	443
1. Getrusage Example .....	445
1. phpversion() example .....	446
1. Setting an Environment Variable .....	447
1. Connect to a Informix database .....	451
1. Closing a Informix connection .....	452
1. Show all rows of the "orders" table as a html table .....	454
2. Insert some values into the "catalog" table .....	454
1. Informix affected rows.....	458
1. Retrieve Informix sqlca.sqlerrd[x] values.....	459
1. Informix fetch rows .....	460
1. Informix results as HTML table .....	461
1. Fielnames and SQL fieldtypes .....	462

1. Informix SQL fieldproperties .....	463
1. Complete example with authenticated bind.....	485
1. Show the list of attributes held for a particular directory entry .....	493
1. List all values of the "mail" attribute for a directory entry .....	496
1. Produce a list of all organizational units of an organization.....	498
1. LDAP search.....	501
1. Enumerating all LDAP error messages .....	503
1. Generating and catching an error.....	503
A-1. Migration: old start/end tags.....	507
A-2. Migration: first new start/end tags .....	507
A-3. Migration: second new start/end tags .....	507
A-4. Migration: third new start/end tags.....	508
A-5. Migration: old if..endif syntax.....	508
A-6. Migration: new if..endif syntax .....	508
A-7. Migration: old while..endwhile syntax.....	509
A-8. Migration: new while..endwhile syntax .....	509
A-9. Migration from 2.0: return values, old code.....	511
A-10. Migration from 2.0: return values, new code .....	511
A-11. Migration from 2.0: concatenation for strings.....	511
B-1. Fetching function arguments .....	513
B-2. Variable function arguments .....	514
B-3. Checking whether \$foo exists in a symbol table .....	516
B-4. Finding a variable's size in a symbol table.....	516
B-5. Initializing a new array .....	517
B-6. Adding entries to a new array .....	517
B-7. Adding a new resource .....	520
B-8. Using an existing resource.....	521
B-9. Deleting an existing resource.....	521
C-1. Example Debugger Message .....	529

# Preface

PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly.

## About this Manual

This manual is written in SGML using the DocBook DTD (<http://www.ora.com/davenport/>), using DSSSL (<http://www.jclark.com/dsssl/>) (Document Style and Semantics Specification Language) for formatting. The tools used for formatting HTML, TeX and RTF versions are Jade (<http://www.jclark.com/jade/>), written by James Clark (<http://www.jclark.com/bio.htm>) and The Modular DocBook Stylesheets (<http://nwalsh.com/docbook/dsssl/>) written by Norman Walsh (<http://nwalsh.com/>). PHP's documentation framework was assembled by Stig Sæther Bakken (<mailto:stig@php.net>).

# **I. Getting Started**

# Chapter 1. Introduction

## What is PHP?

PHP is a server-side HTML-embedded scripting language.

Simple answer, but what does that mean? An example:

### Example 1-1. An introductory example

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <?php echo "Hi, I'm a PHP script!"; ?>
  </body>
</html>
```

Notice how this is different from a CGI script written in other languages like Perl or C – instead of writing a program with lots of commands to output HTML, you write an HTML script with a some embedded code to do something (in this case, output some text). The PHP code is enclosed in special start and end tags that allow you to jump into and out of PHP mode.

What distinguishes PHP from something like client-side Javascript is that the code is executed on the server. If you were to have a script similar to the above on your server, the client would receive the results of running that script, with no way of determining what the underlying code may be. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve.

## What can PHP do?

At the most basic level, PHP can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies.

Perhaps the strongest and most significant feature in PHP is its support for a wide range of databases. Writing a database-enabled web page is incredibly simple. The following databases are currently supported:

Adabas D	InterBase	Solid
dBase	mSQL	Sybase
Empress	MySQL	Velocis
FilePro	Oracle	Unix dbm
Informix	PostgreSQL	

PHP also has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, or even HTTP. You can also open raw network sockets and interact using other protocols.

## A brief history of PHP

PHP was conceived sometime in the fall of 1994 by Rasmus Lerdorf (<mailto:rasmus@lerdorf.on.ca>). Early non-released versions were used on his home page to keep track of who was looking at his online resume. The first version used by others was available sometime in early 1995 and was known as the Personal Home Page Tools. It consisted of a very simplistic parser engine that only understood a few special macros and a number of utilities that were in common use on home pages back then. A guestbook, a counter and some other stuff. The parser was rewritten in mid-1995 and named PHP/FI Version 2. The FI came from another package Rasmus had written which interpreted html form data. He combined the Personal Home Page tools scripts with the Form Interpreter and added mSQL support and PHP/FI was born. PHP/FI grew at an amazing pace and people started contributing code to it.

It is difficult to give any hard statistics, but it is estimated that by late 1996 PHP/FI was in use on at least 15,000 web sites around the world. By mid-1997 this number had grown to over 50,000. Mid-1997 also saw a change in the development of PHP. It changed from being Rasmus' own pet project that a handful of people had contributed to, to being a much more organized team effort. The parser was rewritten from scratch by Zeev Suraski and Andi Gutmans and this new parser formed the basis for PHP Version 3. A lot of the utility code from PHP/FI was ported over to PHP3 and a lot of it was completely rewritten.

Today (mid-1999) either PHP/FI or PHP3 ships with a number of commercial products such as C2's StrongHold web server and RedHat Linux. A conservative estimate based on an extrapolation from numbers provided by NetCraft would be that PHP is in use on over 150,000 sites around the world. To put that in perspective, that is more sites than run Netscape's flagship Enterprise server on the Internet.

Also as of this writing, work is underway on the next generation of PHP, which will utilize the powerful Zend (<http://www.zend.com>) scripting engine to deliver higher performance, and will also support running under web servers other than Apache as a native server module.

# Chapter 2. Installation

## Downloading the latest version

The source code, and binary distributions for some platforms (including Windows), can be found at <http://www.php.net/>.

## Installation on UNIX systems

This section will guide you through the configuration and installation of PHP. Prerequisite knowledge and software:

- Basic UNIX skills (being able to operate "make" and a C compiler)
- An ANSI C compiler
- A web server

## Quick Installation Instructions (Apache Module Version)

```
1. gunzip apache_1.3.x.tar.gz
2. tar xvf apache_1.3.x.tar
3. gunzip php-3.0.x.tar.gz
4. tar xvf php-3.0.x.tar
5. cd apache_1.3.x
6. ./configure --prefix=/www
7. cd ../php-3.0.x
8. ./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars
9. make
10. make install
11. cd ../apache_1.3.x
12. ./configure --prefix=/www --activate-module=src/modules/php3/libphp3.a
13. make
14. make install
```

Instead of this step you may prefer to simply copy the httpd binary

overtop of your existing binary. Make sure you shut down your server first though.

15. `cd ../php-3.0.x`
16. `cp php3.ini-dist /usr/local/lib/php3.ini`

You can edit `/usr/local/lib/php3.ini` file to set PHP options. If you prefer this file in another location, use `-with-config-file-path=/path` in step 8.

17. Edit your `httpd.conf` or `srm.conf` file and add:

```
AddType application/x-httpd-php3 .php3
```

You can choose any extension you wish here. `.php3` is simply the one we suggest.

18. Use your normal procedure for starting the Apache server. (You must stop and restart the server, not just cause the server to reload by use a HUP or USR1 signal.)

## Configuration

There are two ways of configuring PHP.

- Using the "setup" script that comes with PHP. This script asks you a series of questions (almost like the "install" script of PHP/FI 2.0) and runs "configure" in the end. To run this script, type **`./setup`**.

This script will also create a file called "do-conf", this file will contain the options passed to configure. You can edit this file to change just a few options without having to re-run setup. Then type **`./do-conf`** to run configure with the new options.

- Running configure by hand. To see what options you have, type **`./configure --help`**.

Details about some of the different configuration options are listed below.

## Apache module

To build PHP as an Apache module, answer "yes" to "Build as an Apache module?" (the `-with-apache=DIR` option to configure) and specify the Apache distribution base directory. If you have unpacked your Apache distribution in `/usr/local/www/apache_1.2.4`, this is your Apache distribution base directory. The default directory is `/usr/local/etc/httpd`.

## fhttpd module

To build PHP as an fhttpd module, answer "yes" to "Build as an fhttpd module?" (the `-with-fhttpd=DIR` option to configure) and specify the fhttpd source base directory. The default directory is `/usr/local/src/fhttpd`. If you are running fhttpd, building PHP as a module will give better performance, more control and remote execution capability.

## CGI version

The default is to build PHP as a CGI program. If you are running a web server PHP has module support for, you should generally go for that solution for performance reasons. However, the CGI version enables Apache users to run different PHP-enabled pages under different user-ids. Please make sure you read through the Security chapter if you are going to run PHP as a CGI.

## Database Support Options

PHP has native support for a number of databases (as well as ODBC):

### Adabas D

`-with-adabas=DIR`

Compiles with Adabas D support. The parameter is the Adabas D install directory and defaults to `/usr/local/adabasd`.

Adabas home page (<http://www.adabas.com/>)

### dBase

`-with-dbase`

Enables the bundled DBase support. No external libraries are required.

## filePro

```
-with-filepro
```

Enables the bundled read-only filePro support. No external libraries are required.

## mSQL

```
-with-mysql=DIR
```

Enables mSQL support. The parameter to this option is the mSQL install directory and defaults to `/usr/local/Hughes`. This is the default directory of the mSQL 2.0 distribution. **configure** automatically detects which mSQL version you are running and PHP supports both 1.0 and 2.0, but if you compile PHP with mSQL 1.0, you can only access mSQL 1.0 databases, and vice-versa.

See also mSQL Configuration Directives in the configuration file.

mSQL home page (<http://www.hughes.com.au>)

## MySQL

```
-with-mysql=DIR
```

Enables MySQL support. The parameter to this option is the MySQL install directory and defaults to `/usr/local`. This is the default installation directory of the MySQL distribution.

See also MySQL Configuration Directives in the configuration file.

MySQL home page (<http://www.tcx.se>)

## iODBC

```
-with-iodbc=DIR
```

Includes iODBC support. This feature was first developed for iODBC Driver Manager, a freely redistributable ODBC driver manager which runs under many flavors of UNIX. The parameter to this option is the iODBC installation directory and defaults to `/usr/local`.

FreeODBC home page (<http://users.ids.net/~bjepson/freeODBC/>)

## OpenLink ODBC

`-with-openlink=DIR`

Includes OpenLink ODBC support. The parameter to this option is the OpenLink ODBC installation directory and defaults to `/usr/local/openlink`.

OpenLink Software's home page (<http://www.openlinksw.com/>)

## Oracle

`-with-oracle=DIR`

Includes Oracle support. Has been tested and should be working at least with Oracle versions 7.0 through 7.3. The parameter is the `ORACLE_HOME` directory. You do not have to specify this parameter if your Oracle environment has been set up.

Oracle home page (<http://www.oracle.com>)

## PostgreSQL

`-with-pgsql=DIR`

Includes PostgreSQL support. The parameter is the PostgreSQL base install directory and defaults to `/usr/local/pgsql`.

See also Postgres Configuration Directives in the configuration file.

PostgreSQL home page (<http://www.postgreSQL.org/>)

## Solid

`-with-solid=DIR`

Includes Solid support. The parameter is the Solid install directory and defaults to `/usr/local/solid`.

Solid home page (<http://www.solidtech.com>)

## Sybase

`-with-sybase=DIR`

Includes Sybase support. The parameter is the Sybase install directory and defaults to `/home/sybase`.

See also Sybase Configuration Directives in the configuration file.

Sybase home page (<http://www.sybase.com>)

## Sybase-CT

`-with-sybase-ct=DIR`

Includes Sybase-CT support. The parameter is the Sybase-CT install directory and defaults to `/home/sybase`.

See also Sybase-CT Configuration Directives in the configuration file.

## Velocis

`-with-velocis=DIR`

Includes Velocis support. The parameter is the Velocis install directory and defaults to `/usr/local/velocis`.

Velocis home page (<http://www.raima.com>)

## A custom ODBC library

`-with-custom-odbc=DIR`

Includes support for an arbitrary custom ODBC library. The parameter is the base directory and defaults to `/usr/local`.

This option implies that you have defined `CUSTOM_ODBC_LIBS` when you run the configure script. You also must have a valid `odbc.h` header somewhere in your include path. If you don't have one, create it and include your specific header from there. Your header may also require some extra definitions, particularly when it is multiplatform. Define them in `CFLAGS`.

For example, you can use Sybase SQL Anywhere on QNX as following: `CFLAGS=-DODBC_QNX`  
`LDFLAGS=-lnix CUSTOM_ODBC_LIBS="-ldbllib -lodbc" ./configure`  
`-with-custom-odbc=/usr/lib/sqlany50`

## Unified ODBC

`-disable-unified-odbc`

Disables the Unified ODBC module, which is a common interface to all the databases with ODBC-based interfaces, such as Solid and Adabas D. It also works for normal ODBC libraries. Has been tested with iODBC, Solid, Adabas D and Sybase SQL Anywhere. Requires that one (and only one) of these modules or the Velocis module is enabled, or a custom ODBC library specified. This option is only applicable if one of the following options is used: `-with-iodbc`, `-with-solid`, `-with-adabas`, `-with-velocis`, or `-with-custom-odbc`,

See also Unified ODBC Configuration Directives in the configuration file.

## LDAP

`-with-ldap=DIR`

Includes LDAP (Lightweight Directory Access Protocol) support. The parameter is the LDAP base install directory, defaults to `/usr/local/ldap`.

More information about LDAP can be found in RFC1777 (<ftp://ftp.isi.edu/in-notes/rfc1777.txt>) and RFC1778 (<ftp://ftp.isi.edu/in-notes/rfc1778.txt>).

## Other configure options

### **`--with-mcrypt=DIR`**

`--with-mcrypt`

Include support for the mcrypt library. See the mcrypt documentation for more information. If you use the optional *DIR* argument, PHP will look for mcrypt.h in *DIR*/include.

### **`--enable-sysvsem`**

`--enable-sysvsem`

Include support for Sys V semaphores (supported by most Unix derivatives). See the Semaphore and Shared Memory documentation for more information.

### **`--enable-sysvshm`**

`--enable-sysvshm`

Include support for Sys V shared memory (supported by most Unix derivatives). See the Semaphore and Shared Memory documentation for more information.

### **`--with-xml`**

`--with-xml`

Include support for a non-validating XML parser using James Clark's expat library (<http://www.jclark.com/xml/>). See the XML function reference for details.

### **`--enable-maintainer-mode`**

`--enable-maintainer-mode`

Turns on extra dependencies and compiler warnings used by some of the PHP developers.

### **–with-system-regex**

```
-with-system-regex
```

Uses the system’s regular expression library rather than the bundled one. If you are building PHP as a server module, you must use the same library when building PHP as when linking the server. Enable this if the system’s library provides special features you need. It is recommended that you use the bundled library if possible.

### **–with-config-file-path**

```
-with-config-file-path=DIR
```

The path used to look for the configuration file when PHP starts up.

### **–with-exec-dir**

```
-with-exec-dir=DIR
```

Only allow running of executables in DIR when in safe mode. Defaults to `/usr/local/bin`. This option only sets the default, it may be changed with the `safe_mode_exec_dir` directive in the configuration file later.

### **–enable-debug**

```
-enable-debug
```

Enables extra debug information. This makes it possible to gather more detailed information when there are problems with PHP. (Note that this doesn’t have anything to do with debugging facilities or information available to PHP scripts.)

### **—enable-safe-mode**

```
-enable-safe-mode
```

Enables "safe mode" by default. This imposes several restrictions on what PHP can do, such as opening only files within the document root. Read the Security chapter for more more information. CGI users should always enable secure mode. This option only sets the default, it may be enabled or disabled with the `safe_mode` directive in the configuration file later.

### **—enable-track-vars**

```
-enable-track-vars
```

Makes PHP keep track of where GET/POST/cookie variables come from in the arrays `HTTP_GET_VARS`, `HTTP_POST_VARS` and `HTTP_COOKIE_VARS`. This option only sets the default, it may be enabled or disabled with the `track_vars` directive in the configuration file later.

### **—enable-magic-quotes**

```
-enable-magic-quotes
```

Enable magic quotes by default. This option only sets the default, it may be enabled or disabled with the `magic_quotes_runtime` directive in the configuration file later. See also the `magic_quotes_gpc` and the `magic_quotes_sybase` directives.

### **—enable-debugger**

```
-enable-debugger
```

Enables the internal PHP debugger support. This feature is still in an experimental state. See also the Debugger Configuration directives in the configuration file.

### **—enable-discard-path**

```
-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. Read the section in the security chapter about this option.

### **—enable-bcmath**

```
-enable-bcmath
```

Enables **bc** style arbitrary precision math functions. See also the `bcmath.scale` option in the configuration file.

### **—enable-force-cgi-redirect**

```
-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.

When using PHP as a CGI binary, PHP by default always first checks that it is used by redirection (for example under Apache, by using Action directives). This makes sure that the PHP binary cannot be used to bypass standard web server authentication procedures by calling it directly, like `http://my.host/cgi-bin/php/secret/doc.html`. This example accesses `http://my.host/secret/doc.html` but does not honour any security settings enforced by httpd for directory `/secret`.

Not enabling option disables the check and enables bypassing httpd security and authentication settings. Do this only if your server software is unable to indicate that a safe redirection was done and all your files under your document root and user directories may be accessed by anyone.

Read the section in the security chapter about this option.

### **—disable-short-tags**

```
-disable-short-tags
```

Disables the short form `<? ?>` PHP tags. You must disable the short form if you want to use PHP with XML. With short tags disabled, the only PHP code tag is `<?php ?>`. This option only sets the default, it

may be enabled or disabled with the `short_open_tag` directive in the configuration file later.

### **—enable-url-includes**

```
-enable-url-includes
```

Makes it possible to run code on other HTTP or FTP servers directly from PHP with `include()`. See also the `include_path` option in the configuration file.

### **—disable-syntax-hl**

```
-disable-syntax-hl
```

Turns off syntax highlighting.

## **CPPFLAGS and LDFLAGS**

To make the PHP installation look for header or library files in different directories, modify the `CPPFLAGS` and `LDFLAGS` environment variables, respectively. If you are using a sensible shell, you should be able to do **`LDFLAGS=-L/my/lib/dir CPPFLAGS=-I/my/include/dir ./configure`**

## **Building**

When PHP is configured, you are ready to build the CGI executable or the PHP library. The command **`make`** should take care of this. If it fails and you can't figure out why, see the Problems section.

## **Testing**

If you have built PHP as a CGI program, you may test your build by typing **`make test`**. It is always a good idea to test your build. This way you may catch a problem with PHP on your platform early instead of having to struggle with it later.

## Benchmarking

If you have built PHP as a CGI program, you may benchmark your build by typing **make bench**. Note that if safe mode is on by default, the benchmark may not be able to finish if it takes longer than the 30 seconds allowed. This is because the `set_time_limit` can not be used in safe mode. Use the `max_execution_time` configuration setting to control this time for your own scripts. **make bench** ignores the configuration file.

## Installation on Windows 95/98/NT systems

This install guide will help you install and configure PHP on your Windows 9x/NT web servers. This guide was compiled by Bob Silva (mailto:bob\_silva@mail.umesd.k12.or.us). The latest revision can be found at <http://www.umesd.k12.or.us/php/win32install.html>.

This guide provides installation support for:

- Personal Web Server (Newest version recommended)
- Internet Information Server 3 or 4
- Apache 1.3.x
- Omni HTTPd 2.0b1

## General Installation Steps

The following steps should be performed on all installations before the server specific instructions.

- Extract the distribution file to a directory of your choice. "C:\PHP3\" is a good start.
- Copy the file, 'php3-dist.ini' to your '%WINDOWS%' directory and rename it to 'php3.ini'. Your '%WINDOWS%' directory is typically:  
c:\windows for Windows 95/98  
c:\winnt or c:\winnt40 for NT servers
- Edit your 'php3.ini' file:
  - You will need to change the 'extension\_dir' setting to point to your php-install-dir, or where you have placed your 'php3\_\*.dll' files. ex: c:\php3

- If you are using Omni Httpd, do not follow the next step. Set the 'doc\_root' to point to your webserver's document\_root. ex: c:\apache\htdocs or c:\webroot
- Choose which modules you would like to load when PHP starts. You can uncomment the: 'extension=php3\_\*.dll' lines to load these modules. Some modules require you to have additional libraries installed on your system for the module to work correctly. The PHP FAQ (<http://www.php.net/FAQ.php3>) has more information on where to get supporting libraries. You can also load a module dynamically in your script using: **dl("php\_\*.dll");**
- On PWS and IIS, you can set the browscap.ini to point to: 'c:\windows\system\inetsrv\browscap.ini' on Windows 95/98 and 'c:\winnt\system32\inetsrv\browscap.ini' on NT Server. Additional information on using the browscap functionality in PHP can be found at this mirror (<http://www.netvision.net.il/browser-id.php3>), select the "source" button to see it in action.

The DLLs for PHP extensions are prefixed with 'php3\_'. This prevents confusion between PHP extensions and their supporting libraries.

## Windows 95/98/NT and PWS/IIS 3

The recommended method for configuring these servers is to use the INF file included with the distribution (php\_iis\_reg.inf). You may want to edit this file and make sure the extensions and PHP install directories match your configuration. Or you can follow the steps below to do it manually.

WARNING: These steps involve working directly with the windows registry. One error here can leave your system in an unstable state. We highly recommend that you back up your registry first. The PHP Development team will not be held responsible if you damage your registry.

- Run Regedit.
- Navigate to: HKEY\_LOCAL\_MACHINE /System /CurrentControlSet /Services /W3Svc /Parameters /ScriptMap.
- On the edit menu select: New->String Value.
- Type in the extension you wish to use for your php scripts. ex: .php3
- Double click on the new string value and enter the path to php.exe in the value data field. ex: c:\php3\php.exe %s %s. The '%s %s' is VERY important, PHP will not work properly without it.
- Repeat these steps for each extension you wish to associate with PHP scripts.
- Now navigate to: HKEY\_CLASSES\_ROOT
- On the edit menu select: New->Key.

- Name the key to the extension you setup in the previous section. ex: `.php3`
- Highlight the new key and in the right side pane, double click the "default value" and enter `phpfile`.
- Repeat the last step for each extension you set up in the previous section.
- Now create another New->Key under `HKEY_CLASSES_ROOT` and name it `phpfile`.
- Highlight the new key `phpfile` and in the right side pane, double click the "default value" and enter `PHP Script`.
- Right click on the `phpfile` key and select New->Key, name it `Shell`.
- Right click on the `Shell` key and select New->Key, name it `open`.
- Right click on the `open` key and select New->Key, name it `command`.
- Highlight the new key `command` and in the right side pane, double click the "default value" and enter the path to `php.exe`. ex: `c:\php3\php.exe -q %1`. (don't forget the `%1`).
- Exit Regedit.

PWS and IIS 3 users now have a fully operational system. IIS 3 users can use a nifty tool (<http://www.genusa.com/iis/iiscfg.html>) from Steven Genusa to configure their script maps.

## Windows NT and IIS 4

To install PHP on an NT Server running IIS 4, follow these instructions:

- In Internet Service Manager (MMC), select the Web site or the starting point directory of an application.
- Open the directory's property sheets (by right clicking and selecting properties), and then click the Home Directory, Virtual Directory, or Directory tab.
- Click the Configuration button, and then click the App Mappings tab.
- Click Add, and in the Executable box, type: `c:\path-to-php-dir\php.exe %s %s`. You MUST have the `%s %s` on the end, PHP will not function properly if you fail to do this.
- In the Extension box, type the file name extension you want associated with PHP scripts. (You must repeat step 5 and 6 for each extension you want associated with PHP scripts. (`.php3` and `.phtml` are common))
- Set up the appropriate security. (This is done in Internet Service Manager), and if your NT Server uses NTFS file system, add execute rights for `IUSR_` to the directory that contains `php.exe`.

## Windows 9x/NT and Apache 1.3.x

You must edit your `srm.conf` or `httpd.conf` to configure Apache to work with the PHP CGI binary.

Although there can be a few variations of configuring PHP under Apache, this one is simple enough to be used by the newcomer. Please consult the Apache Docs for further configuration directives.

- `ScriptAlias /php3/ "c:/path-to-php-dir/"`
- `AddType application/x-httpd-php3 .php3`
- `AddType application/x-httpd-php3 .phtml`
- `Action application/x-httpd-php3 "/php3/php.exe"`

To use the source code highlighting feature, simply create a PHP script file and stick this code in: `<?php show_source ("original_php_script.php3"); ?>`. Substitute `original_php_script.php3` with the name of the file you wish to show the source of. (this is only one way of doing it). *Note:* On Win-Apache all back slashes in a path statement such as: `"c:\directory\file.ext"`, must be converted to forward slashes.

## Omni HTTPd 2.0b1 for Windows

This has got to be the easiest config there is:

Step 1: Install Omni server

Step 2: Right click on the blue OmniHTTPd icon in the system tray and select **Properties**

Step 3: Click on **Web Server Global Settings**

Step 4: On the 'External' tab, enter: `virtual = .php3 | actual = c:\path-to-php-dir\php.exe`

Step 5: On the **Mime** tab, enter: `virtual = wwwserver/stdcgi | actual = .php3`

Step 6: Click **OK**

Repeat steps 2 - 6 for each extension you want to associate with PHP.

## PHP Modules

**Table 2-1. PHP Modules**

<code>php3_calendar.dll</code>	Calendar conversion functions
<code>php3_crypt.dll</code>	Crypt functions

php3_dbase.dll	DBase functions
php3_dbm.dll	GDBM emulation via Berkely DB2 library
php3_filepro.dll	READ ONLY access to filepro databases
php3_gd.dll	GD Library functions for gif manipulation
php3_hyperwave.dll	HyperWave functions
php3_imap4r2.dll	IMAP 4 functions
php3_ldap.dll	LDAP functions
php3_msql1.dll	mSQL 1 client
php3_msql2.dll	mSQL 2 client
php3_mssql.dll	MSSQL client (requires MSSQL DB-Libraries)
php3_mysql.dll	MySQL functions
php3_nsmail.dll	Netscape mail functions
php3_oci73.dll	Oracle functions
php3_snmp.dll	SNMP get and walk functions (NT only!)
php3_zlib.dll	ZLib functions

## Problems?

### Read the FAQ

Some problems are more common than others. The most common ones are listed in the PHP FAQ, found at <http://www.php.net/FAQ.php3>

### Bug reports

If you think you have found a bug in PHP, please report it. The PHP developers probably don't know about it, and unless you report it, chances are it won't be fixed. You can report bugs using the bug-tracking system at <http://www.php.net/bugs.php3>.

## **Other problems**

If you are still stuck, someone on the PHP mailing list may be able to help you. You should check out the archive first, in case someone already answered someone else who had the same problem as you. The archives are available from the support page on <http://www.php.net/>. To subscribe to the PHP mailing list, send an empty mail to `php3-subscribe@lists.php.net` (`mailto:php3-subscribe@lists.php.net`). The mailing list address is `php3@lists.php.net`.

If you want to get help on the mailing list, please try to be precise and give the necessary details about your environment (which operating system, what PHP version, what web server, if you are running PHP as CGI or a server module, etc.), and preferably enough code to make others able to reproduce and test your problem.

# Chapter 3. Configuration

## The configuration file

The configuration file (called `php3.ini` in PHP 3.0, and simply `php.ini` as of PHP 4.0) is read when PHP starts up. For the server module versions of PHP, this happens only once when the web server is started. For the CGI version, it happens on every invocation.

When using PHP as an Apache module, you can also change the configuration settings using directives in Apache configuration files and `.htaccess` files.

With PHP 3.0, there are Apache directives that correspond to each configuration setting in the `php3.ini` name, except the name is prefixed by "php3\_".

With PHP 4.0, there are just a few Apache directives that allow you to change the PHP configuration settings.

`php_value name value`

This sets the value of the specified variable.

`php_flag name on/off`

This is used to set a Boolean configuration option.

`php_admin_value name value`

This sets the value of the specified variable. "Admin" configuration settings can only be set from within the main Apache configuration files, and not from `.htaccess` files.

`php_admin_flag name on/off`

This is used to set a Boolean configuration option.

You can view the settings of the configuration values in the output of `phpinfo`. You can also access the values of individual configuration settings using `get_cfg_var`.

## General Configuration Directives

*asp\_tags* boolean

Enables the use of ASP-like `<% %>` tags in addition to the usual `<?php ?>` tags. This includes the variable-value printing shorthand of `<%= $value %>`. For more information, see Escaping from HTML.

**Note:** Support for ASP-style tags was added in 3.0.4.

*auto\_append\_file* string

Specifies the name of a file that is automatically parsed after the main file. The file is included as if it was called with the `include` function, so `include_path` is used.

The special value `none` disables auto-appending.

**Note:** If the script is terminated with `exit`, auto-append will *not* occur.

*auto\_prepend\_file* string

Specifies the name of a file that is automatically parsed before the main file. The file is included as if it was called with the `include` function, so `include_path` is used.

The special value `none` disables auto-prepend.

*cgi\_ext* string

*display\_errors* boolean

This determines whether errors should be printed to the screen as part of the HTML output or not.

*doc\_root* string

PHP's "root directory" on the server. Only used if non-empty. If PHP is configured with safe mode, no files outside this directory are served.

*engine* boolean

This directive is really only useful in the Apache module version of PHP. It is used by sites that would like to turn PHP parsing on and off on a per-directory or per-virtual server basis. By putting **php3\_engine off** in the appropriate places in the `httpd.conf` file, PHP can be enabled or disabled.

*error\_log* string

Name of file where script errors should be logged. If the special value `syslog` is used, the errors are sent to the system logger instead. On UNIX, this means `syslog(3)` and on Windows NT it means the event log. The system logger is not supported on Windows 95.

*error\_reporting* integer

Set the error reporting level. The parameter is an integer representing a bit field. Add the values of the error reporting levels you want.

**Table 3-1. Error Reporting Levels**

bit value	enabled reporting
1	normal errors
2	normal warnings
4	parser errors
8	non-critical style-related warnings

The default value for this directive is 7 (normal errors, normal warnings and parser errors are shown).

*open\_basedir* string

Limit the files that can be opened by PHP to the specified directory-tree.

When a script tries to open a file with, for example, `fopen` or `gzopen`, the location of the file is checked. When the file is outside the specified directory-tree, PHP will refuse to open it. All symbolic links are resolved, so it's not possible to avoid this restriction with a symlink.

The special value `.` indicates that the directory in which the script is stored will be used as base-directory.

Under Windows, separate the directories with a semicolon. On all other systems, separate the directories with a colon. As an Apache module, `open_basedir` paths from parent directories are now automatically inherited.

**Note:** Support for multiple directories was added in 3.0.7.

The default is to allow all files to be opened.

*gpc\_order* string

Set the order of GET/POST/COOKIE variable parsing. The default setting of this directive is "GPC". Setting this to "GP", for example, will cause PHP to completely ignore cookies and to overwrite any GET method variables with POST-method variables of the same name.

*ignore\_user\_abort* string

Off by default. If changed to On scripts will run to completion even if the remote client disconnects in the middle. See also *ignore\_user\_abort*.

*include\_path* string

Specifies a list of directories where the *require*, *include* and *fopen\_with\_path* functions look for files. The format is like the system's PATH environment variable: a list of directories separated with a colon in UNIX or semicolon in Windows.

#### **Example 3-1. UNIX *include\_path***

```
include_path=.: /home/httpd/php-lib
```

#### **Example 3-2. Windows *include\_path***

```
include_path=".;c:\www\phplib"
```

The default value for this directive is *.* (only the current directory).

*isapi\_ext* string

*log\_errors* boolean

Tells whether script error messages should be logged to the server's error log. This option is thus server-specific.

*magic\_quotes\_gpc* boolean

Sets the *magic\_quotes* state for GPC (Get/Post/Cookie) operations. When *magic\_quotes* are on, all ' (single-quote), " (double quote), \ (backslash) and NUL's are escaped with a backslash

automatically. If `magic_quotes_sybase` is also on, a single-quote is escaped with a single-quote instead of a backslash.

`magic_quotes_runtime` boolean

If `magic_quotes_runtime` is enabled, most functions that return data from any sort of external source including databases and text files will have quotes escaped with a backslash. If `magic_quotes_sybase` is also on, a single-quote is escaped with a single-quote instead of a backslash.

`magic_quotes_sybase` boolean

If `magic_quotes_sybase` is also on, a single-quote is escaped with a single-quote instead of a backslash if `magic_quotes_gpc` or `magic_quotes_runtime` is enabled.

`max_execution_time` integer

This sets the maximum time in seconds a script is allowed to take before it is terminated by the parser. This helps prevent poorly written scripts from tying up the server.

`memory_limit` integer

This sets the maximum amount of memory in bytes that a script is allowed to allocate. This helps prevent poorly written scripts for eating up all available memory on a server.

`nsapi_ext` string

`short_open_tag` boolean

Tells whether the short form (`<? ?>`) of PHP's open tag should be allowed. If you want to use PHP in combination with XML, you have to disable this option. If disabled, you must use the long form of the open tag (`<?php ?>`).

`sql.safe_mode` boolean

`track_errors` boolean

If enabled, the last error message will always be present in the global variable `$php_errormsg`.

*track\_vars* boolean

If enabled, GET, POST and cookie input can be found in the global associative arrays `$HTTP_GET_VARS`, `$HTTP_POST_VARS` and `$HTTP_COOKIE_VARS`, respectively.

*upload\_tmp\_dir* string

The temporary directory used for storing files when doing file upload. Must be writable by whatever user PHP is running as.

*user\_dir* string

The base name of the directory used on a user's home directory for PHP files, for example `public_html`.

*warn\_plus\_overloading* boolean

If enabled, this option makes PHP output a warning when the plus (+) operator is used on strings. This is to make it easier to find scripts that need to be rewritten to using the string concatenator instead (.

## Mail Configuration Directives

*SMTP* string

DNS name or IP address of the SMTP server PHP under Windows should use for mail sent with the `mail` function.

*sendmail\_from* string

Which "From:" mail address should be used in mail sent from PHP under Windows.

*sendmail\_path* string

Where the **sendmail** program can be found, usually `/usr/sbin/sendmail` or `/usr/lib/sendmail` **configure** does an honest attempt of locating this one for you and set a default, but if it fails, you can set it here.

Systems not using sendmail should set this directive to the sendmail wrapper/replacement their mail system offers, if any. For example, Qmail (<http://www.qmail.org/>) users can normally set it to `/var/qmail/bin/sendmail`.

## Safe Mode Configuration Directives

*safe\_mode* boolean

Whether to enable PHP's safe mode. Read the Security chapter for more more information.

*safe\_mode\_exec\_dir* string

If PHP is used in safe mode, `system` and the other functions executing system programs refuse to start programs that are not in this directory.

## Debugger Configuration Directives

*debugger.host* string

DNS name or IP address of host used by the debugger.

*debugger.port* string

Port number used by the debugger.

*debugger.enabled* boolean

Whether the debugger is enabled.

## Extension Loading Directives

*enable\_dl* boolean

This directive is really only useful in the Apache module version of PHP. You can turn dynamic loading of PHP extensions with `dl` on and off per virtual server or per directory.

The main reason for turning dynamic loading off is security. With dynamic loading, it's possible to ignore all the `safe_mode` and `open_basedir` restrictions.

The default is to allow dynamic loading, except when using safe-mode. In safe-mode, it's always impossible to use `dl`.

*extension\_dir* string

In what directory PHP should look for dynamically loadable extensions.

*extension* string

Which dynamically loadable extensions to load when PHP starts up.

## MySQL Configuration Directives

*mysql.allow\_persistent* boolean

Whether to allow persistent MySQL connections.

*mysql.max\_persistent* integer

The maximum number of persistent MySQL connections per process.

*mysql.max\_links* integer

The maximum number of MySQL connections per process, including persistent connections.

## mSQL Configuration Directives

*mssql.allow\_persistent* boolean

Whether to allow persistent mSQL connections.

*mssql.max\_persistent* integer

The maximum number of persistent mSQL connections per process.

*mssql.max\_links* integer

The maximum number of mSQL connections per process, including persistent connections.

## Postgres Configuration Directives

*pgsql.allow\_persistent* boolean

Whether to allow persistent Postgres connections.

*pgsql.max\_persistent* integer

The maximum number of persistent Postgres connections per process.

*pgsql.max\_links* integer

The maximum number of Postgres connections per process, including persistent connections.

## Sybase Configuration Directives

*sybase.allow\_persistent* boolean

Whether to allow persistent Sybase connections.

*sybase.max\_persistent* integer

The maximum number of persistent Sybase connections per process.

*sybase.max\_links* integer

The maximum number of Sybase connections per process, including persistent connections.

## Sybase-CT Configuration Directives

*sybct.allow\_persistent* boolean

Whether to allow persistent Sybase-CT connections. The default is on.

*sybct.max\_persistent* integer

The maximum number of persistent Sybase-CT connections per process. The default is -1 meaning unlimited.

*sybct.max\_links* integer

The maximum number of Sybase-CT connections per process, including persistent connections. The default is -1 meaning unlimited.

*sybct.min\_server\_severity* integer

Server messages with severity greater than or equal to *sybct.min\_server\_severity* will be reported as warnings. This value can also be set from a script by calling *sybase\_min\_server\_severity*. The default is 10 which reports errors of information severity or greater.

*sybct.min\_client\_severity* integer

Client library messages with severity greater than or equal to *sybct.min\_client\_severity* will be reported as warnings. This value can also be set from a script by calling *sybase\_min\_client\_severity*. The default is 10 which effectively disables reporting.

*sybct.login\_timeout* integer

The maximum time in seconds to wait for a connection attempt to succeed before returning failure. Note that if *max\_execution\_time* has been exceeded when a connection attempt times out, your script will be terminated before it can take action on failure. The default is one minute.

*sybct.timeout* integer

The maximum time in seconds to wait for a *select\_db* or query operation to succeed before returning failure. Note that if *max\_execution\_time* has been exceeded when an operation times out, your script will be terminated before it can take action on failure. The default is no limit.

*sybct.hostname* string

The name of the host you claim to be connecting from, for display by *sp\_who*. The default is none.

## Informix Configuration Directives

*ifx.allow\_persistent* boolean

Whether to allow persistent Informix connections.

*ifx.max\_persistent* integer

The maximum number of persistent Informix connections per process.

*ifx.max\_links* integer

The maximum number of Informix connections per process, including persistent connections.

*ifx.default\_host* string

The default host to connect to when no host is specified in *ifx\_connect* or *ifx\_pconnect*.

*ifx.default\_user* string

The default user id to use when none is specified in *ifx\_connect* or *ifx\_pconnect*.

*ifx.default\_password* string

The default password to use when none is specified in *ifx\_connect* or *ifx\_pconnect*.

*ifx.blobinfile* boolean

Set to true if you want to return blob columns in a file, false if you want them in memory. You can override the setting at runtime with *ifx\_blobinfile\_mode*.

*ifx.textasvarchar* boolean

Set to true if you want to return TEXT columns as normal strings in select statements, false if you want to use blob id parameters. You can override the setting at runtime with *ifx\_textasvarchar*.

*ifx.byteasvarchar* boolean

Set to true if you want to return BYTE columns as normal strings in select queries, false if you want to use blob id parameters. You can override the setting at runtime with *ifx\_textasvarchar*.

*ifx.charasvarchar* boolean

Set to true if you want to trim trailing spaces from CHAR columns when fetching them.

*ifx.nullformat* boolean

Set to true if you want to return NULL columns as the literal string "NULL", false if you want them returned as the empty string "". You can override this setting at runtime with *ifx\_nullformat*.

## BC Math Configuration Directives

*bcmath.scale* integer

Number of decimal digits for all bcmath functions.

## Browser Capability Configuration Directives

*browscap* string

Name of browser capabilities file. See also `get_browser`.

## Unified ODBC Configuration Directives

*uodbc.default\_db* string

ODBC data source to use if none is specified in `odbc_connect` or `odbc_pconnect`.

*uodbc.default\_user* string

User name to use if none is specified in `odbc_connect` or `odbc_pconnect`.

*uodbc.default\_pw* string

Password to use if none is specified in `odbc_connect` or `odbc_pconnect`.

*uodbc.allow\_persistent* boolean

Whether to allow persistent ODBC connections.

*uodbc.max\_persistent* integer

The maximum number of persistent ODBC connections per process.

*uodbc.max\_links* integer

The maximum number of ODBC connections per process, including persistent connections.

# Chapter 4. Security

PHP is a powerful language and the interpreter, whether included in a web server as a module or executed as a separate CGI binary, is able to access files, execute commands and open network connections on the server. These properties make anything run on a web server insecure by default. PHP is designed specifically to be a more secure language for writing CGI programs than Perl or C, and with correct selection of compile-time and runtime configuration options it gives you exactly the combination of freedom and security you need.

As there are many different ways of utilizing PHP, there are many configuration options controlling its behaviour. A large selection of options guarantees you can use PHP for a lot of purposes, but it also means there are combinations of these options and server configurations that result in an insecure setup. This chapter explains the different configuration option combinations and the situations they can be safely used.

## CGI binary

### Possible attacks

Using PHP as a CGI binary is an option for setups that for some reason do not wish to integrate PHP as a module into server software (like Apache), or will use PHP with different kinds of CGI wrappers to create safe chroot and setuid environments for scripts. This setup usually involves installing executable PHP binary to the web server cgi-bin directory. CERT advisory CA-96.11 ([http://www.cert.org/advisories/CA-96.11.interpreters\\_in\\_cgi\\_bin\\_dir.html](http://www.cert.org/advisories/CA-96.11.interpreters_in_cgi_bin_dir.html)) recommends against placing any interpreters into cgi-bin. Even if the PHP binary can be used as a standalone interpreter, PHP is designed to prevent the attacks this setup makes possible:

- Accessing system files: `http://my.host/cgi-bin/php?/etc/passwd`

The query information in a url after the question mark (?) is passed as command line arguments to the interpreter by the CGI interface. Usually interpreters open and execute the file specified as the first argument on the command line.

When invoked as a CGI binary, PHP refuses to interpret the command line arguments.

- Accessing any web document on server: `http://my.host/cgi-bin/php/secret/doc.html`

The path information part of the url after the PHP binary name, `/secret/doc.html` is conventionally used to specify the name of the file to be opened and interpreted by the CGI program. Usually some web server configuration directives (Apache: Action) are used to redirect requests to

documents like `http://my.host/secret/script.php3` to the PHP interpreter. With this setup, the web server first checks the access permissions to the directory `/secret`, and after that creates the redirected request `http://my.host/cgi-bin/php/secret/script.php3`. Unfortunately, if the request is originally given in this form, no access checks are made by web server for file `/secret/script.php3`, but only for the `/cgi-bin/php` file. This way any user able to access `/cgi-bin/php` is able to access any protected document on the web server.

In PHP, compile-time configuration option `--enable-force-cgi-redirect` and runtime configuration directives `doc_root` and `user_dir` can be used to prevent this attack, if the server document tree has any directories with access restrictions. See below for full the explanation of the different combinations.

## Case 1: only public files served

If your server does not have any content that is not restricted by password or ip based access control, there is no need for these configuration options. If your web server does not allow you to do redirects, or the server does not have a way to communicate to the PHP binary that the request is a safely redirected request, you can specify the option `--disable-force-cgi-redirect` to the configure script. You still have to make sure your PHP scripts do not rely on one or another way of calling the script, neither by directly `http://my.host/cgi-bin/php/dir/script.php3` nor by redirection `http://my.host/dir/script.php3`.

Redirection can be configured in Apache by using `AddHandler` and `Action` directives (see below).

## Case 2: using `--enable-force-cgi-redirect`

This compile-time option prevents anyone from calling PHP directly with a url like `http://my.host/cgi-bin/php/secret/dir/script.php3`. Instead, PHP will only parse in this mode if it has gone through a web server redirect rule.

Usually the redirection in the Apache configuration is done with the following directives:

```
Action php3-script /cgi-bin/php
AddHandler php3-script .php3
```

This option has only been tested with the Apache web server, and relies on Apache to set the non-standard CGI environment variable `REDIRECT_STATUS` on redirected requests. If your web server does not support any way of telling if the request is direct or redirected, you cannot use this option and you must use one of the other ways of running the CGI version documented here.

### Case 3: setting `doc_root` or `user_dir`

To include active content, like scripts and executables, in the web server document directories is sometimes consider an insecure practice. If, because of some configuration mistake, the scripts are not executed but displayed as regular HTML documents, this may result in leakage of intellectual property or security information like passwords. Therefore many sysadmins will prefer setting up another directory structure for scripts that are accessible only through the PHP CGI, and therefore always interpreted and not displayed as such.

Also if the method for making sure the requests are not redirected, as described in the previous section, is not available, it is necessary to set up a script `doc_root` that is different from web document root.

You can set the PHP script document root by the configuration directive `doc_root` in the configuration file, or you can set the environment variable `PHP_DOCUMENT_ROOT`. If it is set, the CGI version of PHP will always construct the file name to open with this `doc_root` and the path information in the request, so you can be sure no script is executed outside this directory (except for `user_dir` below).

Another option usable here is `user_dir`. When `user_dir` is unset, only thing controlling the opened file name is `doc_root`. Opening an url like `http://my.host/~user/doc.php3` does not result in opening a file under users home directory, but a file called `~user/doc.php3` under `doc_root` (yes, a directory name starting with a tilde [`~`]).

If `user_dir` is set to for example `public_php`, a request like `http://my.host/~user/doc.php3` will open a file called `doc.php3` under the directory named `public_php` under the home directory of the user. If the home of the user is `/home/user`, the file executed is `/home/user/public_php/doc.php3`.

`user_dir` expansion happens regardless of the `doc_root` setting, so you can control the document root and user directory access separately.

### Case 4: PHP parser outside of web tree

A very secure option is to put the PHP parser binary somewhere outside of the web tree of files. In `/usr/local/bin`, for example. The only real downside to this option is that you will now have to put a line similar to:

```
#!/usr/local/bin/php
```

as the first line of any file containing PHP tags. You will also need to make the file executable. That is, treat it exactly as you would treat any other CGI script written in Perl or sh or any other common scripting language which uses the `#!` shell-escape mechanism for launching itself.

To get PHP to handle `PATH_INFO` and `PATH_TRANSLATED` information correctly with this setup, the php parser should be compiled with the `--enable-discard-path` configure option.

## **Apache module**

When PHP is used as an Apache module it inherits Apache's user permissions (typically those of the "nobody" user).

## **II. Language Reference**

# Chapter 5. Basic syntax

## Escaping from HTML

There are four ways of escaping from HTML and entering "PHP code mode":

### Example 5-1. Ways of escaping from HTML

1. `<? echo ("this is the simplest, an SGML processing instruction\n"); ?>`
2. `<?php echo("if you want to serve XML documents, do like this\n"); ?>`
3. 

```
<script language="php">
    echo ("some editors (like FrontPage) don't
        like processing instructions");
</script>
```
4. `<% echo ("You may optionally use ASP-style tags"); %>`  
`<%= $variable; # This is a shortcut for "<%echo .." %>`

The first way is only available if short tags have been enabled. This can be done via the `short_tags` function, by enabling the `short_open_tag` configuration setting in the PHP config file, or by compiling PHP with the `--enable-short-tags` option to **configure**.

The fourth way is only available if ASP-style tags have been enabled using the `asp_tags` configuration setting.

**Note:** Support for ASP-style tags was added in 3.0.4.

The closing tag for the block will include the immediately trailing newline if one is present.

## Instruction separation

Instructions are separated the same as in C or perl - terminate each statement with a semicolon.

The closing tag (`?>`) also implies the end of the statement, so the following are equivalent:

```
<?php
    echo "This is a test";
?>

<?php echo "This is a test" ?>
```

## Comments

PHP supports 'C', 'C++' and Unix shell-style comments. For example:

```
<?php
    echo "This is a test"; // This is a one-line c++ style comment
    /* This is a multi line comment
       yet another line of comment */
    echo "This is yet another test";
    echo "One Final Test"; # This is shell-style style comment
?>
```

The "one-line" comment styles actually only comment to the end of the line or the current block of PHP code, whichever comes first.

```
<h1>This is an <?# echo "simple";?> example.</h1>
<p>The header above will say 'This is an example'.
```

You should be careful not to nest 'C' style comments, which can happen when commenting out large blocks.

```
<?php
/*
    echo "This is a test"; /* This comment will cause a problem */
*/
?>
```

## Chapter 6. Types

PHP supports the following types:

- integer
- floating-point numbers
- string
- array
- object

The type of a variable is usually not set by the programmer; rather, it is decided at runtime by PHP depending on the context in which that variable is used.

If you would like to force a variable to be converted to a certain type, you may either cast the variable or use the `settype` function on it.

Note that a variable may behave in different manners in certain situations, depending on what type it is at the time. For more information, see the section on Type Juggling.

### Integers

Integers can be specified using any of the following syntaxes:

```
$a = 1234; # decimal number
$a = -123; # a negative number
$a = 0123; # octal number (equivalent to 83 decimal)
$a = 0x12; # hexadecimal number (equivalent to 18 decimal)
```

### Floating point numbers

Floating point numbers ("doubles") can be specified using any of the following syntaxes:

```
$a = 1.234;
$a = 1.2e3;
```

## Strings

Strings can be specified using one of two sets of delimiters.

If the string is enclosed in double-quotes ("), variables within the string will be expanded (subject to some parsing limitations). As in C and Perl, the backslash ("\") character can be used in specifying special characters:

**Table 6-1. Escaped characters**

sequence	meaning
\n	newline
\r	carriage
\t	horizontal tab
\\	backslash
\\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation

You can escape any other character, but a warning will be issued at the highest warning level.

The second way to delimit a string uses the single-quote (') character, which does not do any variable expansion or backslash processing (except for \" and \' so you can insert backslashes and single-quotes in a singly-quoted string).

## String conversion

When a string is evaluated as a numeric value, the resulting value and type are determined as follows.

The string will evaluate as a double if it contains any of the characters '.', 'e', or 'E'. Otherwise, it will evaluate as an integer.

The value is given by the initial portion of the string. If the string starts with valid numeric data, this will be the value used. Otherwise, the value will be 0 (zero). Valid numeric data is an optional sign, followed by one or more digits (optionally containing a decimal point), followed by an optional exponent. The exponent is an 'e' or 'E' followed by one or more digits.

When the first expression is a string, the type of the variable will depend on the second expression.

```

$foo = 1 + "10.5";           // $foo is double (11.5)
$foo = 1 + "-1.3e3";         // $foo is double (-1299)
$foo = 1 + "bob-1.3e3";      // $foo is integer (1)
$foo = 1 + "bob3";           // $foo is integer (1)
$foo = 1 + "10 Small Pigs";   // $foo is integer (11)
$foo = 1 + "10 Little Piggies"; // $foo is integer (11)
$foo = "10.0 pigs " + 1;      // $foo is integer (11)
$foo = "10.0 pigs " + 1.0;    // $foo is double (11)

```

For more information on this conversion, see the Unix manual page for `strtod(3)`.

## Arrays

Arrays actually act like both hash tables (associative arrays) and indexed arrays (vectors).

### Single Dimension Arrays

PHP supports both scalar and associative arrays. In fact, there is no difference between the two. You can create an array using the `list` or `array` functions, or you can explicitly set each array element value.

```

$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;

```

You can also create an array by simply adding values to the array.

```

$a[] = "hello"; // $a[2] == "hello"
$a[] = "world"; // $a[3] == "world"

```

Arrays may be sorted using the `asort`, `arsort`, `ksort`, `rsort`, `sort`, `uasort`, `usort`, and `uksort` functions depending on the type of sort you want.

You can count the number of items in an array using the `count` function.

You can traverse an array using `next` and `prev` functions. Another common way to traverse an array is to use the `each` function.

## Multi-Dimensional Arrays

Multi-dimensional arrays are actually pretty simple. For each dimension of the array, you add another [key] value to the end:

```
$a[1]          = $f;           # one dimensional examples
$a["foo"]     = $f;

$a[1][0]      = $f;           # two dimensional
$a["foo"][2]  = $f;           # (you can mix numeric and associa-
tive indices)
$a[3]["bar"]  = $f;           # (you can mix numeric and associa-
tive indices)

$a["foo"][4]["bar"][0] = $f;   # four dimensional!
```

You can "fill up" multi-dimensional arrays in many ways, but the trickiest one to understand is how to use the `array` command for associative arrays. These two snippets of code fill up the one-dimensional array in the same way:

```
# Example 1:

$a["color"] = "red";
$a["taste"] = "sweet";
$a["shape"] = "round";
$a["name"] = "apple";
$a[3] = 4;

# Example 2:
$a = array(
    "color" => "red",
    "taste" => "sweet",
    "shape" => "round",
    "name"  => "apple",
    3       => 4
);
```

The `array` function can be nested for multi-dimensional arrays:

```
<?
```

```

$a = array(
    "apple" => array(
        "color" => "red",
        "taste" => "sweet",
        "shape" => "round"
    ),
    "orange" => array(
        "color" => "orange",
        "taste" => "sweet",
        "shape" => "round"
    ),
    "banana" => array(
        "color" => "yellow",
        "taste" => "paste-y",
        "shape" => "banana-shaped"
    )
);

echo $a["apple"]["taste"];    # will output "sweet"
?>

```

## Objects

### Object Initialization

To initialize an object, you use the `new` statement to instantiate the object to a variable.

```

class foo {
    function do_foo () {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();

```

## Type juggling

PHP does not require (or support) explicit type definition in variable declaration; a variable's type is determined by the context in which that variable is used. That is to say, if you assign a string value to variable *var*, *var* becomes a string. If you then assign an integer value to *var*, it becomes an integer.

An example of PHP's automatic type conversion is the addition operator '+'. If any of the operands is a double, then all operands are evaluated as doubles, and the result will be a double. Otherwise, the operands will be interpreted as integers, and the result will also be an integer. Note that this does NOT change the types of the operands themselves; the only change is in how the operands are evaluated.

```
$foo = "0"; // $foo is string (ASCII 48)
$foo++;    // $foo is the string "1" (ASCII 49)
$foo += 1; // $foo is now an integer (2)
$foo = $foo + 1.3; // $foo is now a double (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";    // $foo is integer (15)
```

If the last two examples above seem odd, see String conversion.

If you wish to force a variable to be evaluated as a certain type, see the section on Type casting. If you wish to change the type of a variable, see `settype`.

## Type casting

Type casting in PHP works much as it does in C: the name of the desired type is written in parentheses before the variable which is to be cast.

```
$foo = 10; // $foo is an integer
$bar = (double) $foo; // $bar is a double
```

The casts allowed are:

- (int), (integer) - cast to integer
- (real), (double), (float) - cast to double
- (string) - cast to string
- (array) - cast to array
- (object) - cast to object

Note that tabs and spaces are allowed inside the parentheses, so the following are functionally equivalent:

```
$foo = (int) $bar;  
$foo = ( int ) $bar;
```

It may not be obvious exactly what will happen when casting between certain types. For instance, the following should be noted:

# Chapter 7. Variables

## Variable scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
$a = 1; /* global scope */

Function Test () {
    echo $a; /* reference to local scope variable */
}

Test ();
```

This script will not produce any output because the echo statement refers to a local version of the \$a variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function. An example:

```
$a = 1;
$b = 2;

Function Sum () {
    global $a, $b;

    $b = $a + $b;
}

Sum ();
echo $b;
```

The above script will output "3". By declaring \$a and \$b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined `$GLOBALS` array. The previous example can be rewritten as:

```
$a = 1;
$b = 2;

Function Sum () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum ();
echo $b;
```

The `$GLOBALS` array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element.

Another important feature of variable scoping is the *static* variable. A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:

```
Function Test () {
    $a = 0;
    echo $a;
    $a++;
}
```

This function is quite useless since every time it is called it sets `$a` to 0 and prints "0". The `$a++` which increments the variable serves no purpose since as soon as the function exits the `$a` variable disappears. To make a useful counting function which will not lose track of the current count, the `$a` variable is declared static:

```
Function Test () {
    static $a = 0;
    echo $a;
    $a++;
}
```

Now, every time the `Test()` function is called it will print the value of `$a` and increment it.

Static variables are also essential when functions are called recursively. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it

recurse indefinitely. You must make sure you have an adequate way of terminating the recursion. The following simple function recursively counts to 10:

```
Function Test () {
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
```

## Variable variables

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement such as:

```
$a = "hello";
```

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example, *hello*, can be used as the name of a variable by using two dollar signs. ie.

```
$$a = "world";
```

At this point two variables have been defined and stored in the PHP symbol tree: `$a` with contents "hello" and `$hello` with contents "world". Therefore, this statement:

```
echo "$a ${$a}";
```

produces the exact same output as:

```
echo "$a $hello";
```

ie. they both produce: *hello world*.

In order to use variable variables with arrays, you have to resolve an ambiguity problem. That is, if you write `$$a[1]` then the parser needs to know if you meant to use `$a[1]` as a variable, or if you wanted `$$a` as the variable and then the `[1]` index from that variable. The syntax for resolving this ambiguity is: ``${$a[1]}` for the first case and ``${$a}[1]` for the second.

## Variables from outside PHP

### HTML Forms (GET and POST)

When a form is submitted to a PHP script, any variables from that form will be automatically made available to the script by PHP. For instance, consider the following form:

#### Example 7-1. Simple form variable

```
<form action="foo.php3" method="post">
  Name: <input type="text" name="name"><br>
  <input type="submit">
</form>
```

When submitted, PHP will create the variable `$name`, which will contain whatever what entered into the *Name:* field on the form.

PHP also understands arrays in the context of form variables, but only in one dimension. You may, for example, group related variables together, or use this feature to retrieve values from a multiple select input:

#### Example 7-2. More complex form variables

```
<form action="array.php" method="post">
  Name: <input type="text" name="personal[name]"><br>
  Email: <input type="text" name="personal[email]"><br>
  Beer: <br>
  <select multiple name="beer[]">
    <option value="warthog">Warthog
    <option value="guinness">Guinness
  </select>
  <input type="submit">
```

```
</form>
```

If PHP's `track_vars` feature is turned on, either by the `track_vars` configuration setting or the `<?php_track_vars?>` directive, then variables submitted via the POST or GET methods will also be found in the global associative arrays `$HTTP_POST_VARS` and `$HTTP_GET_VARS` as appropriate.

## IMAGE SUBMIT variable names

When submitting a form, it is possible to use an image instead of the standard submit button with a tag like:

```
<input type=image src="image.gif" name="sub">
```

When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two additional variables, `sub_x` and `sub_y`. These contain the coordinates of the user click within the image. The experienced may note that the actual variable names sent by the browser contains a period rather than an underscore, but PHP converts the period to an underscore automatically.

## HTTP Cookies

PHP transparently supports HTTP cookies as defined by Netscape's Spec ([http://www.netscape.com/newsref/std/cookie\\_spec.html](http://www.netscape.com/newsref/std/cookie_spec.html)). Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the `SetCookie` function. Cookies are part of the HTTP header, so the `SetCookie` function must be called before any output is sent to the browser. This is the same restriction as for the `Header` function. Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data.

If you wish to assign multiple values to a single cookie, just add `[]` to the cookie name. For example:

```
SetCookie ("MyCookie[]", "Testing", time()+3600);
```

Note that a cookie will replace a previous cookie by the same name in your browser unless the path or domain is different. So, for a shopping cart application you may want to keep a counter and pass this along, i.e.

**Example 7-3. SetCookie Example**

```
$Count++;
SetCookie ("Count", $Count, time()+3600);
SetCookie ("Cart[$Count]", $item, time()+3600);
```

**Environment variables**

PHP automatically makes environment variables available as normal PHP variables.

```
echo $HOME; /* Shows the HOME environment variable, if set. */
```

Since information coming in via GET, POST and Cookie mechanisms also automatically create PHP variables, it is sometimes best to explicitly read a variable from the environment in order to make sure that you are getting the right version. The `getenv` function can be used for this. You can also set an environment variable with the `putenv` function.

**Determining variable types**

Because PHP determines the types of variables and converts them (generally) as needed, it is not always obvious what type a given variable is at any one time. PHP includes several functions which find out what type a variable is. They are `gettype`, `is_long`, `is_double`, `is_string`, `is_array`, and `is_object`.

## Chapter 8. Constants

PHP defines several constants and provides a mechanism for defining more at run-time. Constants are much like variables, save for the two facts that constants must be defined using the `define` function, and that they cannot later be redefined to another value.

The predefined constants (always available) are:

### `__FILE__`

The name of the script file presently being parsed. If used within a file which has been included or required, then the name of the included file is given, and not the name of the parent file.

### `__LINE__`

The number of the line within the current script file which is being parsed. If used within a file which has been included or required, then the position within the included file is given.

### `PHP_VERSION`

The string representation of the version of the PHP parser presently in use; e.g. '3.0.8-dev'.

### `PHP_OS`

The name of the operating system on which the PHP parser is executing; e.g. 'Linux'.

### `TRUE`

A true value.

### `FALSE`

A false value.

### `E_ERROR`

Denotes an error other than a parsing error from which recovery is not possible.

### `E_WARNING`

Denotes a condition where PHP knows something is wrong, but will continue anyway; these can be caught by the script itself. An example would be an invalid regexp in `ereg`.

**E\_PARSE**

The parser choked on invalid syntax in the script file. Recovery is not possible.

**E\_NOTICE**

Something happened which may or may not be an error. Execution continues. Examples include using an unquoted string as a hash index, or accessing a variable which has not been set.

The `E_*` constants are typically used with the `error_reporting` function for setting the error reporting level.

You can define additional constants using the `define` function.

Note that these are constants, not C-style macros; only valid scalar data may be represented by a constant.

**Example 8-1. Defining Constants**

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
?>
```

**Example 8-2. Using `__FILE__` and `__LINE__`**

```
<?php
function report_error($file, $line, $message) {
    echo "An error occurred in $file on line $line: $message.";
}

report_error(__FILE__, __LINE__, "Something went wrong!");
?>
```

## Chapter 9. Expressions

Expressions are the most important building stones of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

The most basic forms of expressions are constants and variables. When you type " $\$a = 5$ ", you're assigning '5' into  $\$a$ . '5', obviously, has the value 5, or in other words '5' is an expression with the value of 5 (in this case, '5' is an integer constant).

After this assignment, you'd expect  $\$a$ 's value to be 5 as well, so if you wrote  $\$b = \$a$ , you'd expect it to behave just as if you wrote  $\$b = 5$ . In other words,  $\$a$  is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.

Slightly more complex examples for expressions are functions. For instance, consider the following function:

```
function foo () {  
    return 5;  
}
```

Assuming you're familiar with the concept of functions (if you're not, take a look at the chapter about functions), you'd assume that typing  $\$c = \text{foo}()$  is essentially just like writing  $\$c = 5$ , and you're right. Functions are expressions with the value of their return value. Since `foo()` returns 5, the value of the expression '`foo()`' is 5. Usually functions don't just return a static value but compute something.

Of course, values in PHP don't have to be integers, and very often they aren't. PHP supports three scalar value types: integer values, floating point values and string values (scalar values are values that you can't 'break' into smaller pieces, unlike arrays, for instance). PHP also supports two composite (non-scalar) types: arrays and objects. Each of these value types can be assigned into variables or returned from functions.

So far, users of PHP/FI 2 shouldn't feel any change. However, PHP takes expressions much further, in the same way many other languages do. PHP is an expression-oriented language, in the sense that almost everything is an expression. Consider the example we've already dealt with, ' $\$a = 5$ '. It's easy to see that there are two values involved here, the value of the integer constant '5', and the value of  $\$a$  which is being updated to 5 as well. But the truth is that there's one additional value involved here, and that's the value of the assignment itself. The assignment itself evaluates to the assigned value, in this case 5. In practice, it means that ' $\$a = 5$ ', regardless of what it does, is an expression with the value 5. Thus, writing something like ' $\$b = (\$a = 5)$ ' is like writing ' $\$a = 5; \$b = 5;$ ' (a semicolon marks the end of a statement). Since assignments are parsed in a right to left order, you can also write ' $\$b = \$a = 5$ '.

Another good example of expression orientation is pre- and post-increment and decrement. Users of PHP/FI 2 and many other languages may be familiar with the notation of `variable++` and `variable--`. These are increment and decrement operators. In PHP/FI 2, the statement `'$a++'` has no value (is not an expression), and thus you can't assign it or use it in any way. PHP enhances the increment/decrement capabilities by making these expressions as well, like in C. In PHP, like in C, there are two types of increment - pre-increment and post-increment. Both pre-increment and post-increment essentially increment the variable, and the effect on the variable is identical. The difference is with the value of the increment expression. Pre-increment, which is written `'++$variable'`, evaluates to the incremented value (PHP increments the variable before reading its value, thus the name 'pre-increment'). Post-increment, which is written `'$variable++'` evaluates to the original value of `$variable`, before it was incremented (PHP increments the variable after reading its value, thus the name 'post-increment').

A very common type of expressions are comparison expressions. These expressions evaluate to either 0 or 1, meaning FALSE or TRUE (respectively). PHP supports `>` (bigger than), `>=` (bigger than or equal to), `==` (equal), `!=` (not equal), `<` (smaller than) and `<=` (smaller than or equal to). These expressions are most commonly used inside conditional execution, such as `if` statements.

The last example of expressions we'll deal with here is combined operator-assignment expressions. You already know that if you want to increment `$a` by 1, you can simply write `'$a++'` or `'++$a'`. But what if you want to add more than one to it, for instance 3? You could write `'$a++'` multiple times, but this is obviously not a very efficient or comfortable way. A much more common practice is to write `'$a = $a + 3'`. `'$a + 3'` evaluates to the value of `$a` plus 3, and is assigned back into `$a`, which results in incrementing `$a` by 3. In PHP, as in several other languages like C, you can write this in a shorter way, which with time would become clearer and quicker to understand as well. Adding 3 to the current value of `$a` can be written `'$a += 3'`. This means exactly "take the value of `$a`, add 3 to it, and assign it back into `$a`". In addition to being shorter and clearer, this also results in faster execution. The value of `'$a += 3'`, like the value of a regular assignment, is the assigned value. Notice that it is NOT 3, but the combined value of `$a` plus 3 (this is the value that's assigned into `$a`). Any two-place operator can be used in this operator-assignment mode, for example `'$a -= 5'` (subtract 5 from the value of `$a`), `'$b *= 7'` (multiply the value of `$b` by 7), etc.

There is one more expression that may seem odd if you haven't seen it in other languages, the ternary conditional operator:

```
$first ? $second : $third
```

If the value of the first subexpression is true (non-zero), then the second subexpression is evaluated, and that is the result of the conditional expression. Otherwise, the third subexpression is evaluated, and that is the value.

The following example should help you understand pre- and post-increment and expressions in general a bit better:

```

function double($i) {
    return $i*2;
}
$b = $a = 5;          /* assign the value five into the variable $a and $b */
$c = $a++;            /* post-increment, assign original value of $a
                       (5) to $c */
$d = $e = ++$b;       /* pre-increment, assign the incremented value of
                       $b (6) to $d and $e */

/* at this point, both $d and $e are equal to 6 */

$f = double($d++);    /* assign twice the value of $d before
                       the increment, 2*6 = 12 to $f */
$g = double(++$e);    /* assign twice the value of $e after
                       the increment, 2*7 = 14 to $g */
$h = $g += 10;        /* first, $g is incremented by 10 and ends with the
                       value of 24. the value of the assignment (24) is
                       then assigned into $h, and $h ends with the value
                       of 24 as well. */

```

In the beginning of the chapter we said that we'll be describing the various statement types, and as promised, expressions can be statements. However, not every expression is a statement. In this case, a statement has the form of 'expr' ';' that is, an expression followed by a semicolon. In '\$b=\$a=5;', \$a=5 is a valid expression, but it's not a statement by itself. '\$b=\$a=5;' however is a valid statement.

One last thing worth mentioning is the truth value of expressions. In many events, mainly in conditional execution and loops, you're not interested in the specific value of the expression, but only care about whether it means TRUE or FALSE (PHP doesn't have a dedicated boolean type). The truth value of expressions in PHP is calculated in a similar way to perl. Any numeric non-zero numeric value is TRUE, zero is FALSE. Be sure to note that negative values are non-zero and are thus considered TRUE! The empty string and the string "0" are FALSE; all other strings are TRUE. With non-scalar values (arrays and objects) - if the value contains no elements it's considered FALSE, otherwise it's considered TRUE.

PHP provides a full and powerful implementation of expressions, and documenting it entirely goes beyond the scope of this manual. The above examples should give you a good idea about what expressions are and how you can construct useful expressions. Throughout the rest of this manual we'll write *expr* to indicate any valid PHP expression.

# Chapter 10. Operators

## Arithmetic Operators

Remember basic arithmetic from school? These work just like those.

**Table 10-1. Arithmetic Operators**

example	name	result
<code>\$a + \$b</code>	Addition	Sum of \$a and \$b.
<code>\$a - \$b</code>	Subtraction	Remainder of \$b subtracted from \$a.
<code>\$a * \$b</code>	Multiplication	Product of \$a and \$b.
<code>\$a / \$b</code>	Division	Dividend of \$a and \$b.
<code>\$a % \$b</code>	Modulus	Remainder of \$a divided by \$b.

## String Operators

There is only really one string operator – the concatenation operator (".").

```
$a = "Hello ";  
$b = $a . "World!"; // now $b = "Hello World!"
```

## Assignment Operators

The basic assignment operator is "=". Your first inclination might be to think of this as "equal to". Don't. It really means that the the left operand gets set to the value of the expression on the rights (that is, "gets set to").

The value of an assignment expression is the value assigned. That is, the value of "`$a = 3`" is 3. This allows you to do some tricky things:

```
$a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4.
```

In addition to the basic assignment operator, there are "combined operators" for all of the binary arithmetic and string operators that allow you to use a value in an expression and then set its value to the result of that expression. For example:

```
$a = 3;
$a += 5; // sets $a to 8, as if we had said: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // sets $b to "Hello There!", just like $b = $b . "There!";
```

## Bitwise Operators

Bitwise operators allow you to turn specific bits within an integer on or off.

**Table 10-2. Bitwise Operators**

example	name	result
<code>\$a &amp; \$b</code>	And	Bits that are set in both \$a and \$b are set.
<code>\$a   \$b</code>	Or	Bits that are set in either \$a or \$b are set.
<code>\$a ^ \$b</code>	Xor	Bits that are set in \$a or \$b but not both are set.
<code>~ \$a</code>	Not	Bits that are set in \$a are not set, and vice versa.
<code>\$a &lt;&lt; \$b</code>	Shift left	Shift the bits of \$a \$b steps to the left (each step means "multiply by two")
<code>\$a &gt;&gt; \$b</code>	Shift right	Shift the bits of \$a \$b steps to the right (each step means "divide by two")

## Logical Operators

**Table 10-3. Logical Operators**

example	name	result
\$a and \$b	And	True of both \$a and \$b are true.
\$a or \$b	Or	True if either \$a or \$b is true.
\$a xor \$b	Or	True if either \$a or \$b is true, but not both.
! \$a	Not	True if \$a is not true.
\$a && \$b	And	True of both \$a and \$b are true.
\$a    \$b	Or	True if either \$a or \$b is true.

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See below.)

## Comparison Operators

Comparison operators, as their name imply, allow you to compare two values.

**Table 10-4. Comparison Operators**

example	name	result
\$a == \$b	Equal	True if \$a is equal to \$b.
\$a != \$b	Not equal	True if \$a is not equal to \$b.
\$a < \$b	Less than	True if \$a is strictly less than \$b.
\$a > \$b	Greater than	True if \$a is strictly greater than \$b.
\$a <= \$b	Less than or equal to	True if \$a is less than or equal to \$b.
\$a >= \$b	Greater than or equal to	True if \$a is greater than or equal to \$b.

Another conditional operator is the "?:" (or trinary) operator, which operates as in C and many other languages.

```
(expr1) ? (expr2) : (expr3);
```

This expression returns to *expr2* if *expr1* evaluates to true, and *expr3* if *expr1* evaluates to false.

## Operator Precedence

The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression `1 + 5 * 3`, the answer is 16 and not 18 because the multiplication ("`*`") operator has a higher precedence than the addition ("`+`") operator.

The following table lists the precedence of operators with the lowest-precedence operators listed first.

**Table 10-5. Operator Precedence**

Associativity	Operators
left	,
left	or
left	xor
left	and
right	print
left	= += -= *= /= .= %= &= != ~= <<= >>=
left	? :
left	
left	&&
left	
left	^
left	&
non-associative	== !=
non-associative	< <= > >=
left	<< >>
left	+ - .
left	* / %
right	! ~ ++ - (int) (double) (string) (array) (object) @
right	[

Associativity	Operators
non-associative	new

# Chapter 11. Control Structures

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing (an empty statement). Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces. A statement-group is a statement by itself as well. The various statement types are described in this chapter.

## if

The `if` construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code fragments. PHP features an `if` structure that is similar to that of C:

```
if (expr)
    statement
```

As described in the section about expressions, `expr` is evaluated to its truth value. If `expr` evaluates to `TRUE`, PHP will execute `statement`, and if it evaluates to `FALSE` - it'll ignore it.

The following example would display `a is bigger than b` if `$a` is bigger than `$b`:

```
if ($a > $b)
    print "a is bigger than b";
```

Often you'd want to have more than one statement to be executed conditionally. Of course, there's no need to wrap each statement with an `if` clause. Instead, you can group several statements into a statement group. For example, this code would display `a is bigger than b` if `$a` is bigger than `$b`, and would then assign the value of `$a` into `$b`:

```
if ($a > $b) {
    print "a is bigger than b";
    $b = $a;
}
```

If statements can be nested indefinitely within other `if` statements, which provides you with complete flexibility for conditional execution of the various parts of your program.

## else

Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what `else` is for. `else` extends an `if` statement to execute a statement in case the expression in the `if` statement evaluates to `FALSE`. For example, the following code would display `a is bigger than b` if `$a` is bigger than `$b`, and `a is NOT bigger than b` otherwise:

```
if ($a > $b) {
    print "a is bigger than b";
} else {
    print "a is NOT bigger than b";
}
```

The `else` statement is only executed if the `if` expression evaluated to `FALSE`, and if there were any `elseif` expressions - only if they evaluated to `FALSE` as well (see below).

## elseif

`elseif`, as its name suggests, is a combination of `if` and `else`. Like `else`, it extends an `if` statement to execute a different statement in case the original `if` expression evaluates to `FALSE`. However, unlike `else`, it will execute that alternative expression only if the `elseif` conditional expression evaluates to `TRUE`. For example, the following code would display `a is bigger than b`, `a equal to b` or `a is smaller than b`:

```
if ($a > $b) {
    print "a is bigger than b";
} elseif ($a == $b) {
    print "a is equal to b";
} else {
    print "a is smaller than b";
}
```

There may be several `elseif`s within the same `if` statement. The first `elseif` expression (if any) that evaluates to `true` would be executed. In PHP, you can also write `'else if'` (in two words) and the behavior would be identical to the one of `'elseif'` (in a single word). The syntactic meaning is slightly different (if you're familiar with C, this is the same behavior) but the bottom line is that both would result in exactly the same behavior.

The `elseif` statement is only executed if the preceding `if` expression and any preceding `elseif` expressions evaluated to `FALSE`, and the current `elseif` expression evaluated to `TRUE`.

## Alternative syntax for `if` structures: `if () : ... endif;`

PHP offers a different way to group statements within an `if` statement. This is most commonly used when you nest HTML blocks inside `if` statements, but can be used anywhere. Instead of using curly braces, `if (expr)` should be followed by a colon, the list of one or more statements, and end with `endif;`. Consider the following example:

```
<?php if ($a==5): ?>
A = 5
<?php endif; ?>
```

In the above example, the HTML block "A = 5" is nested within an `if` statement written in the alternative syntax. The HTML block would be displayed only if `$a` is equal to 5.

The alternative syntax applies to `else` and `elseif` as well. The following is an `if` structure with `elseif` and `else` in the alternative format:

```
if ($a == 5):
    print "a equals 5";
    print "...";
elseif ($a == 6):
    print "a equals 6";
    print "!!!";
else:
    print "a is neither 5 nor 6";
endif;
```

## while

`while` loops are the simplest type of loop in PHP. They behave just like their C counterparts. The basic

form of a `while` statement is:

```
while (expr) statement
```

The meaning of a `while` statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the `while` expression evaluates to `TRUE`. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration). Sometimes, if the `while` expression evaluates to `FALSE` from the very beginning, the nested statement(s) won't even be run once.

Like with the `if` statement, you can group multiple statements within the same `while` loop by surrounding a group of statements with curly braces, or by using the alternate syntax:

```
while (expr): statement ... endwhile;
```

The following examples are identical, and both print numbers from 1 to 10:

```
/* example 1 */

$i = 1;
while ($i <= 10) {
    print $i++; /* the printed value would be
                  $i before the increment
                  (post-increment) */
}

/* example 2 */

$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

## do..while

`do..while` loops are very similar to `while` loops, except the truth expression is checked at the end of each iteration instead of in the beginning. The main difference from regular `while` loops is that the first iteration of a `do..while` loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it's may not necessarily run with a regular `while` loop (the truth expression is checked at the beginning of each iteration, if it evaluates to `FALSE` right from the beginning, the loop execution would end immediately).

There is just one syntax for `do..while` loops:

```
$i = 0;
do {
    print $i;
} while ($i>0);
```

The above loop would run one time exactly, since after the first iteration, when truth expression is checked, it evaluates to `FALSE` (`$i` is not bigger than 0) and the loop execution ends.

Advanced C users may be familiar with a different usage of the `do..while` loop, to allow stopping execution in the middle of code blocks, by encapsulating them with `do..while(0)`, and using the `break` statement. The following code fragment demonstrates this:

```
do {
    if ($i < 5) {
        print "i is not big enough";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
    print "i is ok";
    ...process i...
} while(0);
```

Don't worry if you don't understand this right away or at all. You can code scripts and even powerful scripts without using this 'feature'.

## for

for loops are the most complex loops in PHP. They behave like their C counterparts. The syntax of a for loop is:

```
for (expr1; expr2; expr3) statement
```

The first expression (*expr1*) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, *expr2* is evaluated. If it evaluates to `TRUE`, the loop continues and the nested statement(s) are executed. If it evaluates to `FALSE`, the execution of the loop ends.

At the end of each iteration, *expr3* is evaluated (executed).

Each of the expressions can be empty. *expr2* being empty means the loop should be run indefinitely (PHP implicitly considers it as `TRUE`, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional `break` statement instead of using the `for` truth expression.

Consider the following examples. All of them display numbers from 1 to 10:

```
/* example 1 */

for ($i = 1; $i <= 10; $i++) {
    print $i;
}

/* example 2 */

for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}

/* example 3 */

$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
```

```

        $i++;
    }

    /* example 4 */

    for ($i = 1; $i <= 10; print $i, $i++) ;

```

Of course, the first example appears to be the nicest one (or perhaps the fourth), but you may find that being able to use empty expressions in `for` loops comes in handy in many occasions.

PHP also supports the alternate "colon syntax" for `for` loops.

```

for (expr1; expr2; expr3): statement; ...; endfor;

```

Other languages have a `foreach` statement to traverse an array or hash. PHP uses the `while` statement and the `list` and `each` functions for this. See the documentation for these functions for an example.

## break

`break` breaks out of the current looping control-structures.

```

$i = 0;
while ($i < 10) {
    if ($arr[$i] == "stop") {
        break;
    }
    $i++;
}

```

## continue

`continue` is used within looping structures to skip the rest of the current loop iteration and continue

execution at the beginning of the next iteration.

```
while (list($key,$value) = each($arr)) {
    if ($key % 2) { // skip even members
        continue;
    }
    do_something_odd ($value);
}
```

## switch

The `switch` statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the `switch` statement is for.

The following two examples are two different ways to write the same thing, one using a series of `if` statements, and the other using the `switch` statement:

```
if ($i == 0) {
    print "i equals 0";
}
if ($i == 1) {
    print "i equals 1";
}
if ($i == 2) {
    print "i equals 2";
}

switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
}
```

It is important to understand how the `switch` statement is executed in order to avoid mistakes. The `switch` statement executes line by line (actually, statement by statement). In the beginning, no code is executed. Only when a `case` statement is found with a value that matches the value of the `switch` expression does PHP begin to execute the statements. PHP continues to execute the statements until the end of the `switch` block, or the first time it sees a `break` statement. If you don't write a `break` statement at the end of a case's statement list, PHP will go on executing the statements of the following case. For example:

```
switch ($i) {
    case 0:
        print "i equals 0";
    case 1:
        print "i equals 1";
    case 2:
        print "i equals 2";
}
```

Here, if `$i` equals to 0, PHP would execute all of the print statements! If `$i` equals to 1, PHP would execute the last two print statements, and only if `$i` equals to 2, you'd get the 'expected' behavior and only 'i equals 2' would be displayed. So, it's important not to forget `break` statements (even though you may want to avoid supplying them on purpose under certain circumstances).

The statement list for a case can also be empty, which simply passes control into the statement list for the next case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i is less than 3 but not negative";
        break;
    case 3:
        print "i is 3";
}
```

A special case is the default case. This case matches anything that wasn't matched by the other cases. For example:

```

switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
    default:
        print "i is not equal to 0, 1 or 2";
}

```

The case expression may be any expression that evaluates to a scalar type, that is, integer or floating-point numbers and strings. Arrays or objects are meaningless in that context.

## require

The `require` statement replaces itself with the specified file, much like the C preprocessor's `#include` works.

This means that you can't put a `require` statement inside of a loop structure and expect it to include the contents of a different file on each iteration. To do that, use an `include` statement.

```
require 'header.inc';
```

## include

The `include` statement includes and evaluates the specified file.

This happens each time the `include` statement is encountered, so you can use an `include` statement within a looping structure to include a number of different file.

```

$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) {
    include $files[$i];
}

```

```
}
```

`include` differs from `require` in that the `include` statement is re-evaluated each time it is encountered (and only when it is being executed), whereas the `require` statement is replaced by the required file when it is first encountered, whether the contents of the file will be evaluated or not (for example, if it is inside an `if` statement whose condition evaluated to false).

Because `include` is a special language construct, you must enclose it within a statement block if it is inside a conditional block.

```
/* This is WRONG and will not work as desired. */

if ($condition)
    include($file);
else
    include($other);

/* This is CORRECT. */

if ($condition) {
    include($file);
} else {
    include($other);
}
```

When the file is evaluated, the parser begins in "HTML-mode" which will output the contents of the file until the first PHP start tag (`<?>`) is encountered.

See also `readfile`, `require`, `virtual`.

# Chapter 12. Functions

## User-defined functions

A function may be defined using syntax such as the following:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {  
    echo "Example function.\n";  
    return $retval;  
}
```

Any valid PHP code may appear inside a function, even other functions and class definitions.

Functions must be defined before they are referenced.

## Returning values

Values are returned by using the optional return statement. Any type may be returned, including lists and objects.

```
function square ($num) {  
    return $num * $num;  
}  
echo square (4);    // outputs '16'.
```

You can't return multiple values from a function, but similar results can be obtained by returning a list.

```
function small_numbers() {  
    return array (0, 1, 2);  
}  
list ($zero, $one, $two) = small_numbers();
```

## Function arguments

Information may be passed to functions via the argument list, which is a comma-delimited list of variables and/or constants.

PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are not supported, but a similar effect may be obtained by passing arrays.

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

## Making arguments be passed by reference

By default, function arguments are passed by value (so that if you change the value of the argument within the function, it does not get changed outside of the function). If you wish to allow a function to modify its arguments, you must pass them by reference.

If you want an argument to a function to always be passed by reference, you can prepend an ampersand (&) to the argument name in the function definition:

```
function add_some_extra(&$string) {
    $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str;    // outputs 'This is a string, and something extra.'
```

If you wish to pass a variable by reference to a function which does not do this by default, you may prepend an ampersand to the argument name in the function call:

```
function foo ($bar) {
    $bar .= ' and something extra.';
}
$str = 'This is a string, ';
foo ($str);
echo $str;    // outputs 'This is a string, '
foo (&$str);
echo $str;    // outputs 'This is a string, and something extra.'
```

## Default argument values

A function may define C++-style default values for scalar arguments as follows:

```
function makecoffee ($type = "cappuccino") {
    return "Making a cup of $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

The output from the above snippet is:

```
Making a cup of cappuccino.
Making a cup of espresso.
```

The default value must be a constant expression, not (for example) a variable or class member.

In PHP 4.0 it's also possible to specify `unset` for default argument. This means that the argument will not be set at all, if a value is not supplied.

Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected. Consider the following code snippet:

```
function makeyogurt ($type = "acidophilus", $flavour) {
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");    // won't work as expected
```

The output of the above example is:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Making a bowl of raspberry .
```

Now, compare the above with this:

```
function makeyogurt ($flavour, $type = "acidophilus") {  
    return "Making a bowl of $type $flavour.\n";  
}  
  
echo makeyogurt ("raspberry");    // works as expected
```

The output of this example is:

```
Making a bowl of acidophilus raspberry.
```

## old\_function

The `old_function` statement allows you to declare a function using a syntax identical to PHP/FI2 (except you must replace 'function' with 'old\_function').

This is a deprecated feature, and should only be used by the PHP/FI2->PHP3 convertor.

### Warning

Functions declared as `old_function` cannot be called from PHP's internal code. Among other things, this means you can't use them in functions such as `usort`, `array_walk`, and `register_shutdown_function`. You can get around this limitation by writing a wrapper function (in normal PHP3 form) to call the `old_function`.

# Chapter 13. Classes and Objects

## class

A class is a collection of variables and functions working with these variables. A class is defined using the following syntax:

```
<?php
class Cart {
    var $items; // Items in our shopping cart

    // Add $num articles of $artnr to the cart

    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
    }

    // Take $num articles of $artnr out of the cart

    function remove_item ($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>
```

This defines a class named Cart that consists of an associative array of articles in the cart and two functions to add and remove items from this cart.

Classes are types, that is, they are blueprints for actual variables. You have to create a variable of the desired type with the new operator.

```
$cart = new Cart;
$cart->add_item("10", 1);
```

This creates an object `$cart` of the class `Cart`. The function `add_item()` of that object is being called to add 1 item of article number 10 to the cart.

Classes can be extensions of other classes. The extended or derived class has all variables and functions of the base class and what you add in the extended definition. This is done using the `extends` keyword.

```
class Named_Cart extends Cart {
    var $owner;

    function set_owner ($name) {
        $this->owner = $name;
    }
}
```

This defines a class `Named_Cart` that has all variables and functions of `Cart` plus an additional variable `$owner` and an additional function `set_owner()`. You create a named cart the usual way and can now set and get the cart's owner. You can still use normal cart functions on named carts:

```
$ncart = new Named_Cart;    // Create a named cart
$ncart->set_owner ("kris"); // Name that cart
print $ncart->owner;        // print the cart owners name
$ncart->add_item ("10", 1); // (inherited functionality from cart)
```

Within functions of a class the variable `$this` means this object. You have to use `$this->something` to access any variable or function named something within your current object.

Constructors are functions in a class that are automatically called when you create a new instance of a class. A function becomes a constructor when it has the same name as the class.

```
class Auto_Cart extends Cart {
    function Auto_Cart () {
        $this->add_item ("10", 1);
    }
}
```

This defines a class `Auto_Cart` that is a `Cart` plus a constructor which initializes the cart with one item of article number "10" each time a new `Auto_Cart` is being made with "new". Constructors can also take arguments and these arguments can be optional, which makes them much more useful.

```
class Constructor_Cart {
    function Constructor_Cart ($item = "10", $num = 1) {
```

```
        $this->add_item ($item, $num);
    }
}

// Shop the same old boring stuff.

$default_cart = new Constructor_Cart;

// Shop for real...

$different_cart = new Constructor_Cart ("20", 17);
```

### Caution

For derived classes, the constructor of the parent class is not automatically called when the derived class's constructor is called.

## III. Features

## Chapter 14. Error handling

There are 4 types of errors and warnings in PHP. They are:

- 1 - Normal Function Errors
- 2 - Normal Warnings
- 4 - Parser Errors
- 8 - Notices (warnings you can ignore but which may imply a bug in your code)

The above 4 numbers are added up to define an error reporting level. The default error reporting level is 7 which is  $1 + 2 + 4$ , or everything except notices. This level can be changed in the `php3.ini` file with the `error_reporting` directive. It can also be set in your Apache `httpd.conf` file with the `php3_error_reporting` directive or lastly it may be set at runtime within a script using the `error_reporting` function.

All PHP expressions can also be called with the "@" prefix, which turns off error reporting for that particular expression. If an error occurred during such an expression and the `track_errors` feature is enabled, you can find the error message in the global variable `$php_errormsg`.

## Chapter 15. Creating GIF images

PHP is not limited to creating just HTML output. It can also be used to create GIF image files, or even more convenient GIF image streams. You will need to compile PHP with the GD library of image functions for this to work.

### Example 15-1. GIF creation with PHP

```
<?php
    Header("Content-type: image/gif");
    $string=implode($argv," ");
    $im = imagecreatefromgif("images/button1.gif");
    $orange = ImageColorAllocate($im, 220, 210, 60);
    $px = (imagesx($im)-7.5*strlen($string))/2;
    ImageString($im,3,$px,9,$string,$orange);
    ImageGif($im);
    ImageDestroy($im);
?>
```

This example would be called from a page with a tag like: `` The above `button.php3` script then takes this "text" string and overlays it on top of a base image which in this case is "images/button1.gif" and outputs the resulting image. This is a very convenient way to avoid having to draw new button images every time you want to change the text of a button. With this method they are dynamically generated.

## Chapter 16. HTTP authentication with PHP

The HTTP Authentication hooks in PHP are only available when it is running as an Apache module and is hence not available in the CGI version. In an Apache module PHP script, it is possible to use the `Header` function to send an "Authentication Required" message to the client browser causing it to pop up a Username/Password input window. Once the user has filled in a username and a password, the URL containing the PHP script will be called again with the variables, `$PHP_AUTH_USER`, `$PHP_AUTH_PW` and `$PHP_AUTH_TYPE` set to the user name, password and authentication type respectively. Only "Basic" authentication is supported at this point. See the `Header` function for more information.

An example script fragment which would force client authentication on a page would be the following:

### Example 16-1. HTTP Authentication example

```
<?php
    if(!isset($PHP_AUTH_USER)) {
        Header("WWW-Authenticate: Basic realm=\"My Realm\"");
        Header("HTTP/1.0 401 Unauthorized");
        echo "Text to send if user hits Cancel button\n";
        exit;
    } else {
        echo "Hello $PHP_AUTH_USER.<P>";
        echo "You entered $PHP_AUTH_PW as your password.<P>";
    }
?>
```

Instead of simply printing out the `$PHP_AUTH_USER` and `$PHP_AUTH_PW`, you would probably want to check the username and password for validity. Perhaps by sending a query to a database, or by looking up the user in a dbm file.

Watch out for buggy Internet Explorer browsers out there. They seem very picky about the order of the headers. Sending the *WWW-Authenticate* header before the HTTP/1.0 401 header seems to do the trick for now.

In order to prevent someone from writing a script which reveals the password for a page that was authenticated through a traditional external mechanism, the `PHP_AUTH` variables will not be set if external authentication is enabled for that particular page. In this case, the `$REMOTE_USER` variable can be used to identify the externally-authenticated user.

Note, however, that the above does not prevent someone who controls a non-authenticated URL from stealing passwords from authenticated URLs on the same server.

Both Netscape and Internet Explorer will clear the local browser window's authentication cache for the realm upon receiving a server response of 401. This can effectively "log out" a user, forcing them to re-enter their username and password. Some people use this to "time out" logins, or provide a "log-out" button.

### Example 16-2. HTTP Authentication example forcing a new name/password

```
<?php
function authenticate() {
    Header( "WWW-
authenticate: basic realm='Test Authentication System'");
    Header( "HTTP/1.0 401 Unauthorized");
    echo "You must enter a valid login ID and password to ac-
cess this resource\n";
    exit;
}

if(!isset($PHP_AUTH_USER) || ($SeenBefore == 1 && !str-
cmp($OldAuth, $PHP_AUTH_USER)) ) {
    authenticate();
}
else {
    echo "Welcome: $PHP_AUTH_USER<BR>";
    echo "Old: $OldAuth";
    echo "<FORM ACTION=\"$_PHP_SELF\" METHOD=POST>\n";
    echo "<INPUT TYPE=HIDDEN NAME=\"SeenBefore\" VALUE=\"1\">\n";
    echo "<INPUT TYPE=HIDDEN NAME=\"OldAuth\" VALUE=\"$_PHP_AUTH_USER\">\n";
    echo "<INPUT TYPE=Submit VALUE=\"Re Authenticate\">\n";
    echo "</FORM>\n";
}
?>
```

This behavior is not required by the HTTP Basic authentication standard, so you should never depend on this. Testing with Lynx has shown that Lynx does not clear the authentication credentials with a 401 server response, so pressing back and then forward again will open the resource (as long as the credential requirements haven't changed).

Also note that this does not work using Microsoft's IIS server and the CGI version of PHP due to a limitation of IIS.

## Chapter 17. Cookies

PHP transparently supports HTTP cookies. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the `setcookie` function. Cookies are part of the HTTP header, so `setcookie` must be called before any output is sent to the browser. This is the same limitation that `header` has.

Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data. If you wish to assign multiple values to a single cookie, just add `[]` to the cookie name. For more details see the `setcookie` function.

# Chapter 18. Handling file uploads

## POST method uploads

PHP is capable of receiving file uploads from any RFC-1867 compliant browser (which includes Netscape Navigator 3 or later, Microsoft Internet Explorer 3 with a patch from Microsoft, or later without a patch). This feature lets people upload both text and binary files. With PHP's authentication and file manipulation functions, you have full control over who is allowed to upload and what is to be done with the file once it has been uploaded.

Note that PHP also supports PUT-method file uploads as used by Netscape Composer and W3C's Amaya clients. See the PUT Method Support for more details.

A file upload screen can be built by creating a special form which looks something like this:

### Example 18-1. File Upload Form

```
<FORM ENCTYPE="multipart/form-data" ACTION="_URL_" METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="1000">
Send this file: <INPUT NAME="userfile" TYPE="file">
<INPUT TYPE="submit" VALUE="Send File">
</FORM>
```

The `_URL_` should point to a PHP file. The `MAX_FILE_SIZE` hidden field must precede the file input field and its value is the maximum filesize accepted. The value is in bytes. In this destination file, the following variables will be defined upon a successful upload:

- `$userfile` - The temporary filename in which the uploaded file was stored on the server machine.
- `$userfile_name` - The original name of the file on the sender's system.
- `$userfile_size` - The size of the uploaded file in bytes.
- `$userfile_type` - The mime type of the file if the browser provided this information. An example would be "image/gif".

Note that the `"$userfile"` part of the above variables is whatever the name of the `INPUT` field of `TYPE=file` is in the upload form. In the above upload form example, we chose to call it "userfile".

Files will by default be stored in the server's default temporary directory. This can be changed by setting the environment variable `TMPDIR` in the environment in which PHP runs. Setting it using `putenv` from within a PHP script will not work.

The PHP script which receives the uploaded file should implement whatever logic is necessary for determining what should be done with the uploaded file. You can for example use the `$file_size` variable to throw away any files that are either too small or too big. You could use the `$file_type` variable to throw away any files that didn't match a certain type criteria. Whatever the logic, you should either delete the file from the temporary directory or move it elsewhere.

The file will be deleted from the temporary directory at the end of the request if it has not been moved away or renamed.

## Common Pitfalls

The `MAX_FILE_SIZE` item cannot specify a file size greater than the file size that has been set in the `upload_max_filesize` in the `PHP3.ini` file or the corresponding `php3_upload_max_filesize` Apache `.conf` directive. The default is 2 Megabytes.

Please note that the CERN httpd seems to strip off everything starting at the first whitespace in the content-type mime header it gets from the client. As long as this is the case, CERN httpd will not support the file upload feature.

## Uploading multiple files

It is possible to upload multiple files simultaneously and have the information organized automatically in arrays for you. To do so, you need to use the same array submission syntax in the HTML form as you do with multiple selects and checkboxes:

**Note:** Support for multiple file uploads was added in version 3.0.10.

### Example 18-2. Uploading multiple forms

```
<form action="file-upload.html" method="post" enctype="multipart/form-data">
  Send these files:<br>
  <input name="userfile[]" type="file"><br>
  <input name="userfile[]" type="file"><br>
  <input type="submit" value="Send files">
</form>
```

When the above form is submitted, the arrays `$userfile`, `$userfile_name`, and `$userfile_size` will be formed in the global scope (as well as in `$HTTP_POST_VARS`). Each of these will be a numerically indexed array of the appropriate values for the submitted files.

For instance, assume that the filenames `/home/test/review.html` and `/home/test/xwp.out` are submitted. In this case, `$userfile_name[0]` would contain the value `review.html`, and `$userfile_name[1]` would contain the value `xwp.out`. Similarly, `$userfile_size[0]` would contain `review.html`'s filesize, and so forth.

## PUT method support

PHP provides support for the HTTP PUT method used by clients such as Netscape Composer and W3C Amaya. PUT requests are much simpler than a file upload and they look something like this:

```
PUT /path/filename.html HTTP/1.1
```

This would normally mean that the remote client would like to save the content that follows as: `/path/filename.html` in your web tree. It is obviously not a good idea for Apache or PHP to automatically let everybody overwrite any files in your web tree. So, to handle such a request you have to first tell your web server that you want a certain PHP script to handle the request. In Apache you do this with the *Script* directive. It can be placed almost anywhere in your Apache configuration file. A common place is inside a `<Directory>` block or perhaps inside a `<Virtualhost>` block. A line like this would do the trick:

```
Script PUT /put.php3
```

This tells Apache to send all PUT requests for URIs that match the context in which you put this line to the `put.php3` script. This assumes, of course, that you have PHP enabled for the `.php3` extension and PHP is active.

Inside your `put.php3` file you would then do something like this:

```
<? copy($PHP_UPLOADED_FILE_NAME,$DOCUMENT_ROOT.$REQUEST_URI); ?>
```

This would copy the file to the location requested by the remote client. You would probably want to perform some checks and/or authenticate the user before performing this file copy. The only trick here is that when PHP sees a PUT-method request it stores the uploaded file in a temporary file just like those

handled by the POST-method. When the request ends, this temporary file is deleted. So, your PUT handling PHP script has to copy that file somewhere. The filename of this temporary file is in the `$PHP_PUT_FILENAME` variable, and you can see the suggested destination filename in the `$REQUEST_URI` (may vary on non-Apache web servers). This destination filename is the one that the remote client specified. You do not have to listen to this client. You could, for example, copy all uploaded files to a special uploads directory.

## Chapter 19. Using remote files

As long as support for the "URL fopen wrapper" is enabled when you configure PHP (which it is unless you explicitly pass the `-disable-url-fopen-wrapper` flag to configure), you can use HTTP and FTP URLs with most functions that take a filename as a parameter, including the `require` and `include` statements.

**Note:** You can't use remote files in `include` and `require` statements on Windows.

For example, you can use this to open a file on a remote web server, parse the output for the data you want, and then use that data in a database query, or simply to output it in a style matching the rest of your website.

### Example 19-1. Getting the title of a remote page

```
<?php
$file = fopen("http://www.php.net/", "r");
if (!$file) {
    echo "<p>Unable to open remote file.\n";
    exit;
}
while (!feof($file)) {
    $line = fgets($file, 1024);
    /* This only works if the title and its tags are on one line. */
    if (eregi("<title>(.*?)</title>", $line, $out)) {
        $title = $out[1];
        break;
    }
}
fclose($file);
?>
```

You can also write to files on an FTP as long you connect as a user with the correct access rights, and the file doesn't exist already. To connect as a user other than 'anonymous', you need to specify the username (and possibly password) within the URL, such as 'ftp://user:password@ftp.example.com/path/to/file'. (You can use the same sort of syntax to access files via HTTP when they require Basic authentication.)

**Example 19-2. Storing data on a remote server**

```
<?php
    $file = fopen("ftp://ftp.php.net/incoming/outputfile", "w");
    if (!$file) {
        echo "<p>Unable to open remote file for writing.\n";
        exit;
    }
    /* Write the data here. */
    fputs($file, "$HTTP_USER_AGENT\n");
    fclose($file);
?>
```

**Note:** You might get the idea from the example above to use this technique to write to a remote log, but as mentioned above, you can only write to a new file using the URL `fopen()` wrappers. To do distributed logging like that, you should take a look at `syslog`.

# Chapter 20. Connection handling

**Note:** The following applies to 3.0.7 and later.

Internally in PHP a connection status is maintained. There are 3 possible states:

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

When a PHP script is running normally the NORMAL state, is active. If the remote client disconnects the ABORTED state flag is turned on. A remote client disconnect is usually caused by the user hitting his STOP button. If the PHP-imposed time limit (see `set_time_limit`) is hit, the TIMEOUT state flag is turned on.

You can decide whether or not you want a client disconnect to cause your script to be aborted. Sometimes it is handy to always have your scripts run to completion even if there is no remote browser receiving the output. The default behaviour is however for your script to be aborted when the remote client disconnects. This behaviour can be set via the `ignore_user_abort` php3.ini directive as well as through the corresponding `php3_ignore_user_abort` Apache .conf directive or with the `ignore_user_abort` function. If you do not tell PHP to ignore a user abort and the user aborts, your script will terminate. The one exception is if you have registered a shutdown function using `register_shutdown_function`. With a shutdown function, when the remote user hits his STOP button, the next time your script tries to output something PHP will detect that the connection has been aborted and the shutdown function is called. This shutdown function will also get called at the end of your script terminating normally, so to do something different in case of a client disconnect you can use the `connection_aborted` function. This function will return true if the connection was aborted.

Your script can also be terminated by the built-in script timer. The default timeout is 30 seconds. It can be changed using the `max_execution_time` php3.ini directive or the corresponding `php3_max_execution_time` Apache .conf directive as well as with the `set_time_limit` function. When the timer expires the script will be aborted and as with the above client disconnect case, if a shutdown function has been registered it will be called. Within this shutdown function you can check to see if a timeout caused the shutdown function to be called by calling the `connection_timeout` function. This function will return true if a timeout caused the shutdown function to be called.

One thing to note is that both the ABORTED and the TIMEOUT states can be active at the same time. This is possible if you tell PHP to ignore user aborts. PHP will still note the fact that a user may have broken the connection, but the script will keep running. If it then hits the time limit it will be aborted and

your shutdown function, if any, will be called. At this point you will find that `connection_timeout` and `connection_aborted` return true. You can also check both states in a single call by using the `connection_status`. This function returns a bitfield of the active states. So, if both states are active it would return 3, for example.

## Chapter 21. Persistent database connections

Persistent connections are SQL links that do not close when the execution of your script ends. When a persistent connection is requested, PHP checks if there's already an identical persistent connection (that remained open from earlier) - and if it exists, it uses it. If it does not exist, it creates the link. An 'identical' connection is a connection that was opened to the same host, with the same username and the same password (where applicable).

People who aren't thoroughly familiar with the way web servers work and distribute the load may mistake persistent connects for what they're not. In particular, they do *not* give you an ability to open 'user sessions' on the same SQL link, they do *not* give you an ability to build up a transaction efficiently, and they don't do a whole lot of other things. In fact, to be extremely clear about the subject, persistent connections don't give you *any* functionality that wasn't possible with their non-persistent brothers.

Why?

This has to do with the way web servers work. There are three ways in which your web server can utilize PHP to generate web pages.

The first method is to use PHP as a CGI "wrapper". When run this way, an instance of the PHP interpreter is created and destroyed for every page request (for a PHP page) to your web server. Because it is destroyed after every request, any resources that it acquires (such as a link to an SQL database server) are closed when it is destroyed. In this case, you do not gain anything from trying to use persistent connections - they simply don't persist.

The second, and most popular, method is to run PHP as a module in a multiprocess web server, which currently only includes Apache. A multiprocess server typically has one process (the parent) which coordinates a set of processes (its children) who actually do the work of serving up web pages. When each request comes in from a client, it is handed off to one of the children that is not already serving another client. This means that when the same client makes a second request to the server, it may be serviced by a different child process than the first time. What a persistent connection does for you in this case it make it so each child process only needs to connect to your SQL server the first time that it serves a page that makes use of such a connection. When another page then requires a connection to the SQL server, it can reuse the connection that child established earlier.

The last method is to use PHP as a plug-in for a multithreaded web server. Currently this is only theoretical - PHP does not yet work as a plug-in for any multithreaded web servers. Work is progressing on support for ISAPI, WSAPI, and NSAPI (on Windows), which will all allow PHP to be used as a plug-in on multithreaded servers like Netscape FastTrack, Microsoft's Internet Information Server (IIS), and O'Reilly's WebSite Pro. When this happens, the behavior will be essentially the same as for the multiprocess model described before.

If persistent connections don't have any added functionality, what are they good for?

The answer here is extremely simple – efficiency. Persistent connections are good if the overhead to create a link to your SQL server is high. Whether or not this overhead is really high depends on many factors. Like, what kind of database it is, whether or not it sits on the same computer on which your web server sits, how loaded the machine the SQL server sits on is and so forth. The bottom line is that if that connection overhead is high, persistent connections help you considerably. They cause the child process to simply connect only once for its entire lifespan, instead of every time it processes a page that requires connecting to the SQL server. This means that for every child that opened a persistent connection will have its own open persistent connection to the server. For example, if you had 20 different child processes that ran a script that made a persistent connection to your SQL server, you'd have 20 different connections to the SQL server, one from each child.

An important summary. Persistent connections were designed to have one-to-one mapping to regular connections. That means that you should *always* be able to replace persistent connections with non-persistent connections, and it won't change the way your script behaves. It *may* (and probably will) change the efficiency of the script, but not its behavior!

## **IV. Function Reference**

# I. Adabas D functions

The Adabas D functions are deprecated, you probably want to use the Unified ODBC functions instead.

## ada\_afetch

### Name

`ada_afetch` — fetch a result row into an array

### Description

See `odbc_fetch_into`

## ada\_autocommit

### Name

`ada_autocommit` — toggle autocommit behaviour

### Description

See `odbc_autocommit`.

## ada\_close

### Name

`ada_close` — close a connection to an Adabas D server

### Description

See `odbc_close`.

## **ada\_commit**

### **Name**

`ada_commit` — commit a transaction

### **Description**

See `odbc_commit`

## **ada\_connect**

### **Name**

`ada_connect` — connect to an Adabas D datasource

### **Description**

See `odbc_connect`.

## **ada\_exec**

### **Name**

`ada_exec` — prepare and execute a SQL statement

### **Description**

See `odbc_exec` or `odbc_do`.

## **ada\_fetchrow**

### **Name**

`ada_fetchrow` — fetch a row from a result

### **Description**

See `odbc_fetch_row`.

## **ada\_fieldname**

### **Name**

`ada_fieldname` — get the columnname

### **Description**

See `odbc_field_name`.

## **ada\_fieldnum**

### **Name**

`ada_fieldnum` — get column number

### **Description**

See `odbc_field_num`.

## **ada\_fieldtype**

### **Name**

`ada_fieldtype` — get the datatype of a field

### **Description**

See `odbc_field_type`.

## **ada\_freeresult**

### **Name**

`ada_freeresult` — >free resources associated with a result

### **Description**

See `odbc_free_result`.

## **ada\_numfields**

### **Name**

`ada_numfields` — get the number of columns in a result

### **Description**

See `odbc_num_fields`.

## **ada\_numrows**

### **Name**

`ada_numrows` — number of rows in a result

### **Description**

See `odbc_num_rows`.

## **ada\_result**

### **Name**

`ada_result` — get data from results

### **Description**

See `odbc_result`.

## **ada\_resultall**

### **Name**

`ada_resultall` — print result as HTML table

### **Description**

See `odbc_result_all`.

## **ada\_rollback**

### **Name**

`ada_rollback` — rollback a transaction

### **Description**

See `odbc_rollback`.

## **II. Apache-specific functions**

# apache\_lookup\_uri

## Name

`apache_lookup_uri` — Perform a partial request for the specified URI and return all info about it

## Description

```
class apache_lookup_uri (string filename);
```

This performs a partial request for a URI. It goes just far enough to obtain all the important information about the given resource and returns this information in a class. The properties of the returned class are:

- `status`
- `the_request`
- `status_line`
- `method`
- `content_type`
- `handler`
- `uri`
- `filename`
- `path_info`
- `args`
- `boundary`
- `no_cache`
- `no_local_copy`
- `allowed`
- `send_bodyct`
- `bytes_sent`
- `byterange`
- `clength`
- `unparsed_uri`
- `mtime`

request\_time

**Note:** Note: apache\_lookup\_uri only works when PHP is installed as an Apache module

## apache\_note

### Name

apache\_note — Get and set apache request notes

### Description

```
string apache_note(string note_name, string [note_value]);
```

apache\_note is an Apache-specific function which gets and sets values in a request's notes table. If called with one argument, it returns the current value of note *note\_name*. If called with two arguments, it sets the value of note *note\_name* to *note\_value* and returns the previous value of note *note\_name*.

## getallheaders

### Name

getallheaders — Fetch all HTTP request headers

### Description

```
array getallheaders(void);
```

This function returns an associative array of all the HTTP headers in the current request.

**Note:** You can also get at the value of the common CGI variables by reading them from the environment, which works whether or not you are using PHP as an Apache module. Use `phpinfo` to see a list of all of the environment variables defined this way.

#### Example 1. `getallheaders()` Example

```
$headers = getallheaders();
while (list($header, $value) = each($headers)) {
    echo "$header: $value<br>\n";
}
```

This example will display all the request headers for the current request.

**Note:** `getallheaders` is currently only supported when PHP runs as an Apache module.

## virtual

### Name

`virtual` — Perform an Apache sub-request

### Description

```
int virtual(string filename);
```

`virtual` is an Apache-specific function which is equivalent to `<!--#include virtual...-->` in `mod_include`. It performs an Apache sub-request. It is useful for including CGI scripts or `.shtml` files, or anything else that you would parse through Apache. Note that for a CGI script, the script must generate valid CGI headers. At the minimum that means it must generate a Content-type header. For PHP files, you need to use `include` or `require`; `virtual` cannot be used to include a document which is itself a PHP file..

## **III. Array functions**

# array

## Name

`array` — Create an array

## Description

```
array array(...);
```

Returns an array of the parameters. The parameters can be given an index with the `=>` operator.

**Note:** `array` is a language construct used to represent literal arrays, and not a regular function.

The following example demonstrates how to create a two-dimensional array, how to specify keys for associative arrays, and how to skip-and-continue numeric indices in normal arrays.

### Example 1. `array` example

```
$fruits = array(  
    "fruits" => array("a"=>"orange", "b"=>"banana", "c"=>"apple"),  
    "numbers" => array(1, 2, 3, 4, 5, 6),  
    "holes"   => array("first", 5 => "second", "third")  
);
```

See also: `list`.

# array\_count\_values

## Name

`array_count_values` — Counts all the values of an array

## Description

```
array array_count_values (array input);
```

`array_count_values` returns an array using the values of the *input* array as keys and their frequency in *input* as values.

### Example 1. `array_count_values` example

```
$array = array(1, "hello", 1, "world", "hello");
array_count_values($array);           // returns array(1=>2, "hello"=>2, "world"=>1)
```

**Note:** This function was added in PHP 4.0.

## `array_keys`

### Name

`array_keys` — Return all the keys of an array

## Description

```
array array_keys (array input, mixed [search_value]);
```

`array_keys` returns the keys, numeric and string, from the *input* array.

If the optional *search\_value* is specified, then only the keys for that value are returned. Otherwise, all the keys from the *input* are returned.

### Example 1. `array_keys` example

```
$array = array(0 => 100, "color" => "red");
```

```
array_keys($array);           // returns array(0, "color")

$array = array(1, 100, 2, 100);
array_keys($array, 100); // returns array(0, 2)
```

See also `array_values`.

**Note:** This function was added in PHP 4.0.

## array\_merge

### Name

`array_merge` — Merge two or more arrays

### Description

```
array array_merge(array array1, array array2, [ ... ] );
```

`array_merge` merges the elements of two or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays had the same string keys, then the later value for that key will overwrite previous one. If, however, the arrays have the same numeric key, this does not happen since the values are appended.

#### Example 1. `array_merge` example

```
$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green", "shape" => "trapezoid");
array_merge($array1, $array2);
```

Resulting array will be `array("color" => "green", 2, 4, "a", "b", "shape" => "trapezoid")`.

**Note:** This function was added in PHP 4.0.

## array\_pad

### Name

`array_pad` — Pad array to the specified length with a value

### Description

```
array array_pad(array input, int pad_size, mixed pad_value);
```

`array_pad` returns a copy of the *input* padded to size specified by *pad\_size* with value *pad\_value*. If *pad\_size* is positive then the array is padded on the right, if it's negative then on the left. If the absolute value of *pad\_size* is less than or equal to the length of the *input* then no padding takes place.

#### Example 1. array\_pad example

```
$input = array(12, 10, 9);

$result = array_pad($input, 5, 0);
// result is array(12, 10, 9, 0, 0)

$result = array_pad($input, -7, -1);
// result is array(-1, -1, -1, -1, 12, 10, 9)

$result = array_pad($input, 2, "noop");
// not padded
```

## array\_pop

### Name

`array_pop` — Pop the element off the end of array

### Description

```
mixed array_pop(array array);
```

`array_pop` pops and returns the last value of the *array*, shortening the *array* by one element.

#### Example 1. `array_pop` example

```
$stack = array("orange", "apple", "raspberry");  
$fruit = array_pop($stack);
```

After this, `$stack` has only 2 elements: "orange" and "apple", and `$fruit` has "raspberry".

See also `array_push`, `array_shift`, and `array_unshift`.

**Note:** This function was added in PHP 4.0.

## array\_push

### Name

`array_push` — Push one or more elements onto the end of array

## Description

```
int array_push(array array, mixed var, [...]);
```

`array_push` treats *array* as a stack, and pushes the passed variables onto the end of *array*. The length of *array* increases by the number of variables pushed. Has the same effect as:

```
$array[] = $var;
```

repeated for each *var*.

Returns the new number of elements in the array.

### Example 1. `array_push` example

```
$stack = array(1, 2);  
array_push($stack, "+", 3);
```

This example would result in *\$stack* having 4 elements: 1, 2, "+", and 3.

See also: `array_pop`, `array_shift`, and `array_unshift`.

**Note:** This function was added in PHP 4.0.

## `array_reverse`

### Name

`array_reverse` — Return an array with elements in reverse order

### Description

```
array array_reverse(array array);
```

`array_reverse` takes input *array* and returns a new array with the order of the elements reversed.

#### Example 1. `array_reverse` example

```
$input = array("php", 4.0, array("green", "red"));
$result = array_reverse($input);
```

This makes `$result` have `array(array("green", "red"), 4.0, "php")`.

**Note:** This function was added in PHP 4.0 Beta 3.

## `array_shift`

### Name

`array_shift` — Pop an element of the beginning of array

### Description

```
mixed array_shift (array array);
```

`array_shift` shifts the first value of the *array* off and returns it, shortening the *array* by one element and moving everything down.

#### Example 1. `array_shift` example

```
$args = array("-v", "-f");
$opt = array_shift($args);
```

This would result in `$args` having one element `"-f"` left, and `$opt` being `"-v"`.

See also `array_unshift`, `array_push`, and `array_pop`.

**Note:** This function was added in PHP 4.0.

## array\_slice

### Name

`array_slice` — Extract a slice of the array

### Description

```
array array_slice(array array, int offset, int [length] );
```

`array_slice` returns a sequence of elements from the *array* specified by the *offset* and *length* parameters.

If *offset* is positive, the sequence will start at that offset in the *array*. If *offset* is negative, the sequence will start that far from the end of the *array*.

If *length* is given and is positive, then the sequence will have that many elements in it. If *length* is given and is negative then the sequence will stop that many elements from the end of the array. If it is omitted, then the sequence will have everything from *offset* up until the end of the *array*.

#### Example 1. array\_slice examples

```
$input = array("a", "b", "c", "d", "e");

$output = array_slice($input, 2);           // returns "c", "d", and "e"
$output = array_slice($input, 2, -1);      // returns "c", "d"
$output = array_slice($input, -2, 1);      // returns "d"
$output = array_slice($input, 0, 3);       // returns "a", "b", and "c"
```

See also `array_splice`.

**Note:** This function was added in PHP 4.0.

## array\_splice

### Name

`array_splice` — Remove a portion of the array and replace it with something else

### Description

```
array array_splice(array input, int offset, int [length] , array
[replacement]);
```

`array_splice` removed the elements designated by *offset* and *length* from the *input* array, and replaces them with the elements of the *replacement* array, if supplied.

If *offset* is positive then the start of removed portion is at that offset from the beginning of the *input* array. If *offset* is negative then it starts that far from the end of the *input* array.

If *length* is omitted, removes everything from *offset* to the end of the array. If *length* is specified and is positive, then that many elements will be removed. If *length* is specified and is negative then the end of the removed portion will be that many elements from the end of the array. Tip: to remove everything from *offset* to the end of the array when *replacement* is also specified, use `count($input)` for *length*.

If *replacement* array is specified, then the removed elements are replaced with elements from this array. If *offset* and *length* are such that nothing is removed, then the elements from the *replacement* array are inserted in the place specified by the *offset*. Tip: if the replacement is just one element it is not necessary to put `array()` around it, unless the element is an array itself.

The following equivalences hold:

<code>array_push(\$input, \$x, \$y)</code>	<code>array_splice(\$input, count(\$input), 0, array(\$x, \$y))</code>
<code>array_pop(\$input)</code>	<code>array_splice(\$input, -1)</code>
<code>array_shift(\$input)</code>	<code>array_splice(\$input, 0, 1)</code>
<code>array_unshift(\$input, \$x, \$y)</code>	<code>array_splice(\$input, 0, 0, array(\$x, \$y))</code>
<code>\$a[\$x] = \$y</code>	<code>array_splice(\$input, \$x, 1, \$y)</code>

Returns the array consisting of removed elements.

**Example 1. array\_splice examples**

```

$input = array("red", "green", "blue", "yellow");

array_splice($input, 2);          // $input is now array("red", "green")
array_splice($input, 1, -1);     // $input is now array("red", "yellow")
array_splice($input, 1, count($input), "orange");
                                // $input is now array("red", "orange")
array_splice($input, -1, 1, array("black", "maroon"));
                                // $input is now array("red", "green",
                                //                      "blue", "black", "maroon")

```

See also `array_slice`.

**Note:** This function was added in PHP 4.0.

## array\_unshift

### Name

`array_unshift` — Push one or more elements onto the beginning of array

### Description

```
int array_unshift (array array, mixed var, [...]);
```

`array_unshift` prepends passed elements to the front of the *array*. Note that the list of elements is prepended as a whole, so that the prepended elements stay in the same order.

Returns the new number of elements in the *array*.

**Example 1. array\_unshift example**

```
$queue = array("p1", "p3");
```

```
array_unshift($queue, "p4", "p5", "p6");
```

This would result in `$queue` having 5 elements: "p4", "p5", "p6", "p1", and "p3".

See also `array_shift`, `array_push`, and `array_pop`.

**Note:** This function was added in PHP 4.0.

## array\_values

### Name

`array_values` — Return all the values of an array

### Description

```
array array_values (array input);
```

`array_values` returns all the values from the *input* array.

#### Example 1. array\_values example

```
$array = array("size" => "XL", "color" => "gold");  
array_values($array);    // returns array("XL", "gold")
```

**Note:** This function was added in PHP 4.0.

## array\_walk

### Name

`array_walk` — Apply a user function to every member of an array.

### Description

```
int array_walk(array arr, string func, mixed userdata);
```

Applies the function named by *func* to each element of *arr*. *func* will be passed array value as the first parameter and array key as the second parameter. If *userdata* is supplied, it will be passed as the third parameter to the user function. If *func* requires more than two or three arguments, depending on *userdata*, a warning will be generated each time `array_walk` calls *func*. These warnings may be suppressed by prepending the '@' sign to the `array_walk` call, or by using `error_reporting`.

**Note:** If *func* needs to be working with the actual values of the array, specify that the first parameter of *func* should be passed by reference. Then any changes made to those elements will be made in the array itself.

**Note:** Passing the key and *userdata* to *func* was added in 4.0.

#### Example 1. array\_walk example

```
$fruits = array("d"=>"lemon","a"=>"orange","b"=>"banana","c"=>"apple");

function test_alter( &$item1, $key, $prefix ) {
    $item1 = "$prefix: $item1";
}

function test_print( $item2, $key ) {
    echo "$key. $item2<br>\n";
}

array_walk( $fruits, 'test_print' );
array_walk( $fruits, 'test_alter', 'fruit' );
array_walk( $fruits, 'test_print' );
```

See also `each` and `list`.

## arsort

### Name

`arsort` — Sort an array in reverse order and maintain index association

### Description

```
void arsort(array array);
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

#### Example 1. `arsort` example

```
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
arsort($fruits);
for(reset($fruits); $key = key($fruits); next($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[a] = orange` `fruits[d] = lemon` `fruits[b] = banana` `fruits[c] = apple` The fruits have been sorted in reverse alphabetical order, and the index associated with each element has been maintained.

See also: `asort`, `rsort`, `ksort`, and `sort`.

## asort

### Name

`asort` — Sort an array and maintain index association

### Description

```
void asort(array array);
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

#### Example 1. `asort` example

```
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
asort($fruits);
for(reset($fruits); $key = key($fruits); next($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[c] = apple` `fruits[b] = banana` `fruits[d] = lemon` `fruits[a] = orange` The fruits have been sorted in alphabetical order, and the index associated with each element has been maintained.

See also `arsort`, `rsort`, `ksort`, and `sort`.

## compact

### Name

`compact` — Create array containing variables and their values

## Description

```
array compact (string varname / array varnames, [...]);
```

`compact` takes a variable number of parameters. Each parameter can be either a string containing the name of the variable, or an array of variable names. The array can contain other arrays of variable names inside it; `compact` handles it recursively.

For each of these, `compact` looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key. In short, it does the opposite of `extract`. It returns the output array with all the variables added to it.

### Example 1. `compact` example

```
$city = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";

$location_vars = array("city", "state");

$result = compact("event", $location_vars);
```

After this, `$result` will be `array("event" => "SIGGRAPH", "city" => "San Francisco", "state" => "CA")`.

See also `extract`.

**Note:** This function was added in PHP 4.0.

## count

### Name

`count` — count elements in a variable

## Description

```
int count(mixed var);
```

Returns the number of elements in *var*, which is typically an array (since anything else will have one element).

Returns 1 if the variable is not an array.

Returns 0 if the variable is not set.

### Warning

`count` may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. Use `isset` to test if a variable is set.

See also: `sizeof`, `isset`, and `is_array`.

## current

### Name

`current` — Return the current element in an array

### Description

```
mixed current(array array);
```

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

The `current` function simply returns the array element that's currently being pointed by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list, `current` returns false.

### Warning

If the array contains empty elements (0 or "", the empty string) then this function will return false for these elements as well. This makes it impossible to determine if you are really at the end of the list in such an array using `current`. To properly traverse an array that may contain empty elements, use the `each` function.

See also: `end`, `next`, `prev` and `reset`.

## each

### Name

`each` — Return the next key and value pair from an array

### Description

```
array each(array array);
```

Returns the current key and value pair from the array *array* and advances the array cursor. This pair is returned in a four-element array, with the keys *0*, *1*, *key*, and *value*. Elements *0* and *key* contain the key name of the array element, and *1* and *value* contain the data.

If the internal pointer for the array points past the end of the array contents, `each` returns false.

#### Example 1. `each` examples

```
$foo = array ("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each( $foo );
```

`$bar` now contains the following key/value pairs:

- `0 => 0`
- `1 => 'bob'`
- `key => 0`

- value => 'bob'

```
$foo = array( "Robert" => "Bob", "Seppo" => "Sepi" );
$bar = each( $foo );
```

\$bar now contains the following key/value pairs:

- 0 => 'Robert'
- 1 => 'Bob'
- key => 'Robert'
- value => 'Bob'

each is typically used in conjunction with list to traverse an array; for instance, \$HTTP\_POST\_VARS:

#### **Example 2. Traversing \$HTTP\_POST\_VARS with each**

```
echo "Values submitted via POST method:<br>";
while (list($key, $val) = each($HTTP_POST_VARS)) {
    echo "$key => $val<br>";
}
```

After each has executed, the array cursor will be left on the next element of the array, or on the last element if it hits the end of the array.

See also key, list, current, reset, next, and prev.

## end

### **Name**

end — Set the internal pointer of an array to its last element

## Description

```
end(array array);
```

`end` advances *array*'s internal pointer to the last element.

See also: `current`, `each`, `end`, `next`, and `reset`.

## extract

### Name

`extract` — Import variables into the symbol table from an array

## Description

```
void extract(array var_array, int [extract_type] , string [prefix] );
```

This function is used to import variables from an array into the current symbol table. It takes associative array *var\_array* and treats keys as variable names and values as variable values. For each key/value pair it will create a variable in the current symbol table, subject to *extract\_type* and *prefix* parameters.

`extract` checks for collisions with existing variables. The way collisions are treated is determined by *extract\_type*. It can be one of the following values:

#### EXTR\_OVERWRITE

If there is a collision, overwrite the existing variable.

#### EXTR\_SKIP

If there is a collision, don't overwrite the existing variable.

#### EXTR\_PREFIX\_SAME

If there is a collision, prefix the new variable with *prefix*.

**EXTR\_PREFIX\_ALL**

Prefix all variables with *prefix*.

If *extract\_type* is not specified, it is assumed to be EXTR\_OVERWRITE.

Note that *prefix* is only required if *extract\_type* is EXTR\_PREFIX\_SAME or EXTR\_PREFIX\_ALL.

*extract* checks each key to see if it constitutes a valid variable name, and if it does only then does it proceed to import it.

A possible use for *extract* is to import into symbol table variables contained in an associative array returned by *wddx\_deserialize*.

**Example 1. extract example**

```
<?
/* Suppose that $var_array is an array returned from
   wddx_deserialize */
$size = "large";
$var_array = array("color" => "blue",
                  "size"  => "medium",
                  "shape" => "sphere");
extract($var_array, EXTR_PREFIX_SAME, "wddx");

print "$color, $size, $shape, $wddx_size\n";

?>
```

The above example will produce:

```
blue, large, sphere, medium
```

The *\$size* wasn't overwritten, because we specified EXTR\_PREFIX\_SAME, which resulted in *\$wddx\_size* being created. If EXTR\_SKIP was specified, then *\$wddx\_size* wouldn't even have been created. EXTR\_OVERWRITE would have caused *\$size* to have value "medium", and EXTR\_PREFIX\_ALL would result in new variables being named *\$wddx\_color*, *\$wddx\_size*, and *\$wddx\_shape*.

## in\_array

### Name

`in_array` — Return true if a value exists in an array

### Description

```
bool in_array(mixed needle, array haystack);
```

Searches *haystack* for *needle* and returns true if it is found in the array, false otherwise.

#### Example 1. in\_array example

```
$os = array("Mac", "NT", "Irix", "Linux");  
if (in_array("Irix", $os))  
    print "Got Irix";
```

**Note:** This function was added in PHP 4.0.

## key

### Name

`key` — Fetch a key from an associative array

### Description

```
mixed key(array array);
```

`key` returns the index element of the current array position.

See also: `current`, `next`

## krsort

### Name

`krsort` — Sort an array by key in reverse order

### Description

```
int krsort(array array);
```

Sorts an array by key in reverse order, maintaining key to data correlations. This is useful mainly for associative arrays.

#### Example 1. `krsort` example

```
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");  
krsort($fruits);  
for(reset($fruits); $key = key($fruits); next($fruits)) {  
    echo "fruits[$key] = ".$fruits[$key]."\n";  
}
```

This example would display: `fruits[d] = lemon` `fruits[c] = apple` `fruits[b] = banana` `fruits[a] = orange`

See also `asort`, `arsort`, `ksort` `sort`, and `rsort`.

## ksort

### Name

`ksort` — Sort an array by key

### Description

```
int ksort(array array);
```

Sorts an array by key, maintaining key to data correlations. This is useful mainly for associative arrays.

#### Example 1. `ksort` example

```
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
ksort($fruits);
for(reset($fruits); $key = key($fruits); next($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[a] = orange` `fruits[b] = banana` `fruits[c] = apple`  
`fruits[d] = lemon`

See also `asort`, `arsort`, `sort`, and `rsort`.

## list

### Name

`list` — Assign variables as if they were an array

### Description

```
void list(...);
```

Like `array`, this is not really a function, but a language construct. `list` is used to assign a list of variables in one operation.

### Example 1. `list` example

```
<table>
<tr>
  <th>Employee name</th>
  <th>Salary</th>
</tr>
<?php

$result = mysql($conn, "SELECT id, name, salary FROM employees");
while (list($id, $name, $salary) = mysql_fetch_row($result)) {
    print(" <tr>\n".
        "   <td><a href=\"info.php3?id=$id\">$name</a></td>\n".
        "   <td>$salary</td>\n".
        " </tr>\n");
}

?></table>
```

See also: `each`, `array`.

## next

### Name

`next` — Advance the internal array pointer of an array

### Description

```
mixed next(array $array);
```

Returns the array element in the next place that's pointed by the internal array pointer, or false if there are no more elements.

`next` behaves like `current`, with one difference. It advances the internal array pointer one place forward before returning the element. That means it returns the next array element and advances the internal array pointer by one. If advancing the internal array pointer results in going beyond the end of the element list, `next` returns false.

### Warning

If the array contains empty elements then this function will return false for these elements as well. To properly traverse an array which may contain empty elements see the `each` function.

See also: `current`, `end` `prev` and `reset`

## pos

### Name

`pos` — Get the current element from an array

### Description

```
mixed pos(array array);
```

This is an alias for `current`.

See also: `end`, `next`, `prev` and `reset`.

## prev

### Name

`prev` — Rewind the internal array pointer

## Description

```
mixed prev(array array);
```

Returns the array element in the previous place that's pointed by the internal array pointer, or false if there are no more elements.

### Warning

If the array contains empty elements then this function will return false for these elements as well. To properly traverse an array which may contain empty elements see the `each` function.

`prev` behaves just like `next`, except it rewinds the internal array pointer one place instead of advancing it.

See also: `current`, `end next` and `reset`

## range

### Name

`range` — Create an array containing a range of integers

### Description

```
array range(int low, int high);
```

`range` returns an array of integers from *low* to *high*, inclusive.

See `shuffle` for an example of its use.

## reset

### Name

`reset` — Set the internal pointer of an array to its first element

### Description

```
mixed reset(array array);
```

`reset` rewinds *array*'s internal pointer to the first element.

`reset` returns the value of the first array element.

See also: `current`, `each`, `next`, `prev`, and `reset`.

## rsort

### Name

`rsort` — Sort an array in reverse order

### Description

```
void rsort(array array);
```

This function sorts an array in reverse order (highest to lowest).

#### Example 1. `rsort` example

```
$fruits = array("lemon","orange","banana","apple");  
rsort($fruits);  
for (reset($fruits); list($key,$value) = each($fruits); ) {  
    echo "fruits[$key] = ", $value, "\n";  
}
```

```
}
```

This example would display: `fruits[0] = orange fruits[1] = lemon fruits[2] = banana fruits[3] = apple` The fruits have been sorted in reverse alphabetical order.

See also: `arsort`, `asort`, `ksort`, `sort`, and `usort`.

## shuffle

### Name

`shuffle` — Shuffle an array

### Description

```
void shuffle(array array);
```

This function shuffles (randomizes the order of the elements in) an array.

#### Example 1. `shuffle` example

```
$numbers = range(1,20);
srand(time());
shuffle($numbers);
while (list($number) = each($numbers)) {
    echo "$number ";
}
```

See also `arsort`, `asort`, `ksort`, `rsort`, `sort` and `usort`.

## sizeof

### Name

`sizeof` — Get the number of elements in an array

### Description

```
int sizeof(array array);
```

Returns the number of elements in the array.

See also: `count`

## sort

### Name

`sort` — Sort an array

### Description

```
void sort(array array);
```

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

#### Example 1. `sort` example

```
$fruits = array("lemon","orange","banana","apple");  
sort($fruits);  
for(reset($fruits); $key = key($fruits); next($fruits)) {  
    echo "fruits[$key] = ".$fruits[$key]."\n";  
}
```

This example would display: `fruits[0] = apple fruits[1] = banana fruits[2] = lemon fruits[3] = orange` The fruits have been sorted in alphabetical order.

See also: `arsort`, `asort`, `ksort`, `rsort`, and `usort`.

## uasort

### Name

`uasort` — Sort an array with a user-defined comparison function and maintain index association

### Description

```
void uasort(array $array, function $cmp_function);
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant. The comparison function is user-defined.

## uksort

### Name

`uksort` — Sort an array by keys using a user-defined comparison function

### Description

```
void uksort(array $array, function $cmp_function);
```

This function will sort the keys of an array using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

#### Example 1. `uksort` example

```
function mycompare($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}
$a = array(4 => "four", 3 => "three", 20 => "twenty", 10 => "ten");
uksort($a, mycompare);
while(list($key, $value) = each($a)) {
    echo "$key: $value\n";
}
```

This example would display: 20: twenty 10: ten 4: four 3: three

See also: `arsort`, `asort`, `uasort`, `ksort`, `rsort`, and `sort`.

## usort

### Name

`usort` — Sort an array by values using a user-defined comparison function

### Description

```
void usort(array array, function cmp_function);
```

This function will sort an array by its values using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

**Example 1. usort example**

```
function cmp($a,$b) {  
    if ($a == $b) return 0;  
    return ($a > $b) ? -1 : 1;  
}  
$a = array(3,2,5,6,1);  
usort($a, cmp);  
while(list($key,$value) = each($a)) {  
    echo "$key: $value\n";  
}
```

This example would display: 0: 6 1: 5 2: 3 3: 2 4: 1

**Note:** Obviously in this trivial case the `rsort` function would be more appropriate.

**Warning**

The underlying quicksort function in some C libraries (such as on Solaris systems) may cause PHP to crash if the comparison function does not return consistent values.

See also: `arsort`, `asort`, `ksort`, `rsort` and `sort`.

## IV. Aspell functions

The `aspell` functions allows you to check the spelling on a word and offer suggestions.

You need the `aspell` library, available from: <http://metalab.unc.edu/kevina/aspell/>

## aspell\_new

### Name

aspell\_new — load a new dictionary

### Description

```
int aspell_new(string master, string personal);
```

aspell\_new opens up a new dictionary and returns the dictionary link identifier for use in other aspell functions.

#### Example 1. aspell\_new

```
$aspell_link=aspell_new("english");
```

## aspell\_check

### Name

aspell\_check — check a word

### Description

```
boolean aspell_check(int dictionary_link, string word);
```

aspell\_check checks the spelling of a word and returns true if the spelling is correct, false if not.

**Example 1. aspell\_check**

```
$aspell_link=aspell_new("english");
if (aspell_check($aspell_link,"testt")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

## aspell\_check-raw

### Name

aspell\_check-raw — check a word without changing its case or trying to trim it

### Description

```
boolean aspell_check_raw(int dictionary_link, string word);
```

aspell\_check\_raw checks the spelling of a word, without changing its case or trying to trim it in any way and returns true if the spelling is correct, false if not.

**Example 1. aspell\_check\_raw**

```
$aspell_link=aspell_new("english");
if (aspell_check_raw($aspell_link,"testt")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

## aspell\_suggest

### Name

aspell\_suggest — suggest spellings of a word

### Description

```
array aspell_suggest (int dictionary_link, string word);
```

aspell\_suggest returns an array of possible spellings for the given word.

#### Example 1. aspell\_suggest

```
$aspell_link=aspell_new("english");

if (!aspell_check($aspell_link,"testt")) {
    $suggestions=aspell_suggest($aspell_link,"testt");

    for($i=0; $i < count($suggestions); $i++) {
        echo "Possible spelling: " . $suggestions[$i] . "<br>";
    }
}
```

## V. Arbitrary precision mathematics functions

These functions are only available if PHP was configured with `-enable-bcmath`.

## bcadd

### Name

bcadd — Add two arbitrary precision numbers.

### Description

```
string bcadd(string left operand, string right operand, int [scale]);
```

Adds the *left operand* to the *right operand* and returns the sum in a string. The optional *scale* parameter is used to set the number of digits after the decimal place in the result.

See also `bcsub`.

## bccomp

### Name

bccomp — Compare two arbitrary precision numbers.

### Description

```
int bccomp(string left operand, string right operand, int [scale]);
```

Compares the *left operand* to the *right operand* and returns the result as an integer. The optional *scale* parameter is used to set the number of digits after the decimal place which will be used in the comparison. The return value is 0 if the two operands are equal. If the *left operand* is larger than the *right operand* the return value is +1 and if the *left operand* is less than the *right operand* the return value is -1.

## bcdiv

### Name

bcdiv — Divide two arbitrary precision numbers.

### Description

```
string bcdiv(string left operand, string right operand, int [scale]);
```

Divides the *left operand* by the *right operand* and returns the result. The optional *scale* sets the number of digits after the decimal place in the result.

See also `bcmul`.

## bcmod

### Name

bcmod — Get modulus of an arbitrary precision number.

### Description

```
string bcmod(string left operand, string modulus);
```

Get the modulus of the *left operand* using *modulus*.

See also `bcdiv`.

## bcmul

### Name

bcmul — Multiply two arbitrary precision number.

### Description

```
string bcmul(string left operand, string right operand, int [scale]);
```

Multiply the *left operand* by the *right operand* and returns the result. The optional *scale* sets the number of digits after the decimal place in the result.

See also `bcddiv`.

## bcpow

### Name

bcpow — Raise an arbitrary precision number to another.

### Description

```
string bcpow(string x, string y, int [scale]);
```

Raise *x* to the power *y*. The *scale* can be used to set the number of digits after the decimal place in the result.

See also `bcsqrt`.

## bcscale

### Name

`bcscale` — Set default scale parameter for all bc math functions.

### Description

```
string bcscale(int scale);
```

This function sets the default scale parameter for all subsequent bc math functions that do not explicitly specify a scale parameter.

## bcsqrt

### Name

`bcsqrt` — Get the square root of an arbitrary precision number.

### Description

```
string bcsqrt(string operand, int scale);
```

Return the square root of the *operand*. The optional *scale* parameter sets the number of digits after the decimal place in the result.

See also `bcpow`.

## bcsub

### Name

bcsub — Subtract one arbitrary precision number from another.

### Description

```
string bcsub(string left operand, string right operand, int [scale]);
```

Subtracts the *right operand* from the *left operand* and returns the result in a string. The optional *scale* parameter is used to set the number of digits after the decimal place in the result.

See also bcadd.

## VI. Calendar functions

The calendar functions are only available if you have compiled the calendar extension in dl/calendar. Read dl/README for instructions on using it.

The calendar extension presents a series of functions to simplify converting between different calendar formats. The intermediary or standard it is based on is the Julian Day Count. The Julian Day Count is a count of days starting way earlier than any date most people would need to track (somewhere around 4000bc). To convert between calendar systems, you must first convert to Julian Day Count, then to the calendar system of your choice. Julian Day Count is very different from the Julian Calendar! For more information on calendar systems visit <http://genealogy.org/~scottlee/cal-overview.html>. Excerpts from this page are included in these instructions, and are in quotes.

## JDToGregorian

### Name

JDToGregorian — Converts Julian Day Count to Gregorian date

### Description

```
string jdtogregorian(int julianday);
```

Converts Julian Day Count to a string containing the Gregorian date in the format of "month/day/year"

## GregorianToJD

### Name

GregorianToJD — Converts a Gregorian date to Julian Day Count

### Description

```
int gregoriantojd(int month, int day, int year);
```

Valid Range for Gregorian Calendar 4714 B.C. to 9999 A.D.

Although this software can handle dates all the way back to 4714 B.C., such use may not be meaningful. The Gregorian calendar was not instituted until October 15, 1582 (or October 5, 1582 in the Julian calendar). Some countries did not accept it until much later. For example, Britain converted in 1752, The USSR in 1918 and Greece in 1923. Most European countries used the Julian calendar prior to the Gregorian.

#### Example 1. Calendar functions

```
<?php
```

```
$jd = GregorianToJD(10,11,1970);  
echo("$jd\n");  
$gregorian = JDToGregorian($jd);  
echo("$gregorian\n");  
?>
```

## JDToJulian

### Name

JDToJulian — Converts a Julian Calendar date to Julian Day Count

### Description

```
string jdtojulian(int julianday);
```

Converts Julian Day Count to a string containing the Julian Calendar Date in the format of "month/day/year".

## JulianToJD

### Name

JulianToJD — Converts a Julian Calendar date to Julian Day Count

### Description

```
int juliantojd(int month, int day, int year);
```

Valid Range for Julian Calendar 4713 B.C. to 9999 A.D.

Although this software can handle dates all the way back to 4713 B.C., such use may not be meaningful. The calendar was created in 46 B.C., but the details did not stabilize until at least 8 A.D., and perhaps as late as the 4th century. Also, the beginning of a year varied from one culture to another - not all accepted January as the first month.

## JDToJewish

### Name

JDToJewish — Converts a Julian Day Count to the Jewish Calendar

### Description

```
string jdtojewish(int julianday);
```

Converts a Julian Day Count to the Jewish Calendar.

## JewishToJD

### Name

JewishToJD — Converts a date in the Jewish Calendar to Julian Day Count

### Description

```
int jewishtojd(int month, int day, int year);
```

**Valid Range** Although this software can handle dates all the way back to the year 1 (3761 B.C.), such use may not be meaningful.

The Jewish calendar has been in use for several thousand years, but in the early days there was no formula to determine the start of a month. A new month was started when the new moon was first observed.

## JDToFrench

### Name

JDToFrench — Converts a Julian Day Count to the French Republican Calendar

### Description

```
string jdtofrench(int month, int day, int year);
```

Converts a Julian Day Count to the French Republican Calendar.

## FrenchToJD

### Name

FrenchToJD — Converts a date from the French Republican Calendar to a Julian Day Count

### Description

```
int frenchtojd(int month, int day, int year);
```

Converts a date from the French Republican Calendar to a Julian Day Count

These routines only convert dates in years 1 through 14 (Gregorian dates 22 September 1792 through 22 September 1806). This more than covers the period when the calendar was in use.

## JDMonthName

### Name

JDMonthName — Returns a month name

### Description

```
string jdmmonthname(int julianday, int mode);
```

Returns a string containing a month name. *mode* tells this function which calendar to convert the Julian Day Count to, and what type of month names are to be returned.

**Table 1. Calendar modes**

Mode	Meaning
0	Gregorian - apreviated
1	Gregorian
2	Julian - apreviated
3	Julian
4	Jewish
5	French Republican

## JDDayOfWeek

### Name

JDDayOfWeek — Returns the day of the week

### Description

```
mixed jddayofweek(int julianday, int mode);
```

Returns the day of the week. Can return a string or an int depending on the mode.

**Table 1. Calendar week modes**

Mode	Meaning
0	returns the day number as an int (0=sunday, 1=monday, etc)
1	returns string containing the day of week (english-gregorian)
2	returns a string containing the abbreviated day of week (english-gregorian)

## easter\_date

### Name

easter\_date — get UNIX timestamp for midnight on Easter of a given year

### Description

```
int easter_date(int year);
```

Returns the UNIX timestamp corresponding to midnight on Easter of the given year. If no year is specified, the current year is assumed.

*Warning:* This function will generate a warning if the year is outside of the range for UNIX timestamps (i.e. before 1970 or after 2037).

#### Example 1. `easter_date` example

```
echo date( "M-d-Y", easter_date(1999) );      /* "Apr-04-1999" */
echo date( "M-d-Y", easter_date(2000) );      /* "Apr-23-2000" */
echo date( "M-d-Y", easter_date(2001) );      /* "Apr-15-2001" */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See `easter_days` for calculating Easter before 1970 or after 2037.

## `easter_days`

### Name

`easter_days` — get number of days after March 21 on which Easter falls for a given year

### Description

```
int easter_days(int year);
```

Returns the number of days after March 21 on which Easter falls for a given year. If no year is specified, the current year is assumed.

This function can be used instead of `easter_date` to calculate Easter for years which fall outside the range of UNIX timestamps (i.e. before 1970 or after 2037).

**Example 1. `easter_days` example**

```
echo easter_days(1999);      /* 14, i.e. April 4 */
echo easter_days(1492);      /* 32, i.e. April 22 */
echo easter_days(1913);      /* 2, i.e. March 23 */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See also `easter_date`.

## VII. ClibPDF functions

ClibPDF allows to create pdf documents with PHP. It is available at FastIO (<http://www.fastio.com>) but is not free software. You should definitely read the licence before you start playing with ClibPDF. If you cannot fulfil the licence agreement consider using pdflib by Thomas Merz, which is also very powerful. ClibPDF functionality and API is similar to Thomas Merz pdflib but ClibPDF is, according to FastIO, faster and creates smaller documents. This may have changed with the new version 2.0 of pdflib. A simple benchmark (the pdfclock.c example from pdflib 2.0 turned into a php script) actually show no difference in speed at all. The file size is also similar if compression is turned off.

This documentation should be read with the ClibPDF manual since it explains much of the library in much more detail. Once you understand the manual of ClibPDF you should be able to start using the library with PHP.

Many functions in the native ClibPDF and the PHP module, as well as in pdflib, have the same name. All functions except for `cpdf_open` take as their first parameter the handle for the document on which the function is to be performed. Currently this handle is not used internally since ClibPDF does not support the creation of several PDF documents at the same time. Actually, you should not even try it, the results are unpredictable. I cannot oversee what the consequences in a multi threaded environment are.

According to the author of ClibPDF this will change in one of the next releases (current version when this was written is 1.10). If you need this functionality use the pdflib module.

One big advantage of ClibPDF over pdflib is the possibility to create the pdf document completely in memory without using temporary files. It also provides the ability to pass coordinates in a predefined unit length. This is a handy feature but can be simulated with `pdf_translate`.

Most of the functions are fairly easy to use. The most difficult part is probably creating a very simple PDF document at all. The following example should help you get started. It creates a document with one page. The page contains the text "Times-Roman" in an outlined 30pt font. The text is underlined.

### Example 1. Simple ClibPDF Example

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
cpdf_set_font($cpdf, "Times-Roman", 30, 4);
cpdf_set_text_rendering($cpdf, 1);
cpdf_text($cpdf, "Times Roman outlined", 50, 750);
cpdf_moveto($cpdf, 50, 740);
cpdf_lineto($cpdf, 330, 740);
cpdf_stroke($cpdf);
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
```

```

cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>

```

The pdflib distribution contains a more complex example which creates a series of pages with an analog clock. Here is that example converted into PHP using the ClibPDF extension:

### Example 2. pdfclock example from pdflib 2.0 distribution

```

<?php
$radius = 200;
$margin = 20;
$pagecount = 40;

$pdf = cpdf_open(0);
cpdf_set_creator($pdf, "pdf_clock.php3");
cpdf_set_title($pdf, "Analog Clock");

while($pagecount- > 0) {
    cpdf_page_init($pdf, $pagecount+1, 0, 2 * ($radius + $margin), 2 * ($radius + $margin), 1.0);

    cpdf_set_page_animation($pdf, 4, 0.5, 0, 0, 0); /* wipe */

    cpdf_translate($pdf, $radius + $margin, $radius + $margin);
    cpdf_save($pdf);
    cpdf_setrgbcolor($pdf, 0.0, 0.0, 1.0);

    /* minute strokes */
    cpdf_setlinewidth($pdf, 2.0);
    for ($alpha = 0; $alpha < 360; $alpha += 6)
    {
        cpdf_rotate($pdf, 6.0);
        cpdf_moveto($pdf, $radius, 0.0);
        cpdf_lineto($pdf, $radius-$margin/3, 0.0);
        cpdf_stroke($pdf);
    }

    cpdf_restore($pdf);
    cpdf_save($pdf);

    /* 5 minute strokes */
    cpdf_setlinewidth($pdf, 3.0);

```

```

for ($alpha = 0; $alpha < 360; $alpha += 30)
{
    cpdf_rotate($pdf, 30.0);
    cpdf_moveto($pdf, $radius, 0.0);
    cpdf_lineto($pdf, $radius-$margin, 0.0);
    cpdf_stroke($pdf);
}

$lttime = getdate();

/* draw hour hand */
cpdf_save($pdf);
cpdf_rotate($pdf, -(($lttime['minutes']/60.0) + $lttime['hours'] -
3.0) * 30.0);
cpdf_moveto($pdf, -$radius/10, -$radius/20);
cpdf_lineto($pdf, $radius/2, 0.0);
cpdf_lineto($pdf, -$radius/10, $radius/20);
cpdf_closepath($pdf);
cpdf_fill($pdf);
cpdf_restore($pdf);

/* draw minute hand */
cpdf_save($pdf);
cpdf_rotate($pdf, -(($lttime['seconds']/60.0) + $lttime['minutes'] -
15.0) * 6.0);
cpdf_moveto($pdf, -$radius/10, -$radius/20);
cpdf_lineto($pdf, $radius * 0.8, 0.0);
cpdf_lineto($pdf, -$radius/10, $radius/20);
cpdf_closepath($pdf);
cpdf_fill($pdf);
cpdf_restore($pdf);

/* draw second hand */
cpdf_setrgbcolor($pdf, 1.0, 0.0, 0.0);
cpdf_setlinewidth($pdf, 2);
cpdf_save($pdf);
cpdf_rotate($pdf, -(($lttime['seconds'] - 15.0) * 6.0));
cpdf_moveto($pdf, -$radius/5, 0.0);
cpdf_lineto($pdf, $radius, 0.0);
cpdf_stroke($pdf);
cpdf_restore($pdf);

/* draw little circle at center */
cpdf_circle($pdf, 0, 0, $radius/30);
cpdf_fill($pdf);

```

```
    cpdf_restore($pdf);

    cpdf_finalize_page($pdf, $pagecount+1);
}

cpdf_finalize($pdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($pdf);
cpdf_close($pdf);
?>
```

## cpdf\_set\_creator

### Name

`cpdf_set_creator` — Sets the creator field in the pdf document

### Description

```
void cpdf_set_creator(string creator);
```

The `cpdf_set_creator` function sets the creator of a pdf document.

See also `cpdf_set_subject`, `cpdf_set_title`, `cpdf_set_keywords`.

## cpdf\_set\_title

### Name

`cpdf_set_title` — Sets the title field of the pdf document

### Description

```
void cpdf_set_title(string title);
```

The `cpdf_set_title` function sets the title of a pdf document.

See also `cpdf_set_subject`, `cpdf_set_creator`, `cpdf_set_keywords`.

## cpdf\_set\_subject

### Name

`cpdf_set_subject` — Sets the subject field of the pdf document

### Description

```
void cpdf_set_subject (string subject);
```

The `cpdf_set_subject` function sets the subject of a pdf document.

See also `cpdf_set_title`, `cpdf_set_creator`, `cpdf_set_keywords`.

## cpdf\_set\_keywords

### Name

`cpdf_set_keywords` — Sets the keywords field of the pdf document

### Description

```
void cpdf_set_keywords (string keywords);
```

The `cpdf_set_keywords` function sets the keywords of a pdf document.

See also `cpdf_set_title`, `cpdf_set_creator`, `cpdf_set_subject`.

## cpdf\_open

### Name

`cpdf_open` — Opens a new pdf document

### Description

```
int cpdf_open(int compression, string filename);
```

The `cpdf_open` function opens a new pdf document. The first parameter turns document compression on if it is unequal to 0. The second optional parameter sets the file in which the document is written. If it is omitted the document is created in memory and can either be written into a file with the `cpdf_save_to_file` or written to standard output with `cpdf_output_buffer`.

**Note:** The return value will be needed in further versions of ClibPDF as the first parameter in all other functions which are writing to the pdf document.

The ClibPDF library takes the filename "-" as a synonym for stdout. If PHP is compiled as an apache module this will not work because the way ClibPDF outputs to stdout does not work with apache. You can solve this problem by skipping the filename and using `cpdf_output_buffer` to output the pdf document.

See also `cpdf_close`, `cpdf_output_buffer`.

## cpdf\_close

### Name

`cpdf_close` — Closes the pdf document

## Description

```
void cpdf_close(int pdf document);
```

The `cpdf_close` function closes the pdf document. This should be the last function even after `cpdf_finalize`, `cpdf_output_buffer` and `cpdf_save_to_file`.

See also `cpdf_open`.

## cpdf\_page\_init

### Name

`cpdf_page_init` — Starts new page

### Description

```
void cpdf_page_init(int pdf document, int page number, int orientation,  
double height, double width, double unit);
```

The `cpdf_page_init` function starts a new page with height *height* and width *width*. The page has number *page number* and orientation *orientation*. *orientation* can be 0 for portrait and 1 for landscape. The last optional parameter *unit* sets the unit for the koordinate system. The value should be the number of postscript points per unit. Since one inch is equal to 72 points, a value of 72 would set the unit to one inch. The default is also 72.

See also `cpdf_set_current_page`.

## cpdf\_finalize\_page

### Name

cpdf\_finalize\_page — Ends page

### Description

```
void cpdf_finalize_page(int pdf document, int page number);
```

The `cpdf_finalize_page` function ends the page with page number *page number*. This function is only for saving memory. A finalized page takes less memory but cannot be modified anymore.

See also `cpdf_page_init`.

## cpdf\_finalize

### Name

cpdf\_finalize — Ends document

### Description

```
void cpdf_finalize(int pdf document);
```

The `cpdf_finalize` function ends the document. You still have to call `cpdf_close`.

See also `cpdf_close`.

## cpdf\_output\_buffer

### Name

`cpdf_output_buffer` — Outputs the pdf document in memory buffer

### Description

```
void cpdf_output_buffer(int pdf document);
```

The `cpdf_output_buffer` function outputs the pdf document to stdout. The document has to be created in memory which is the case if `cpdf_open` has been called with no filename parameter.

See also `cpdf_open`.

## cpdf\_save\_to\_file

### Name

`cpdf_save_to_file` — Writes the pdf document into a file

### Description

```
void cpdf_save_to_file(int pdf document, string filename);
```

The `cpdf_save_to_file` function outputs the pdf document into a file if it has been created in memory. This function is not needed if the pdf document has been open by specifying a filename as a parameter of `cpdf_open`.

See also `cpdf_output_buffer`, `cpdf_open`.

## cpdf\_set\_current\_page

### Name

`cpdf_set_current_page` — Sets current page

### Description

```
void cpdf_set_current_page(int pdf document, int page number);
```

The `cpdf_set_current_page` function set the page on which all operations are performed. One can switch between pages until a page is finished with `cpdf_finalize_page`.

See also `cpdf_finalize_page`.

## cpdf\_begin\_text

### Name

`cpdf_begin_text` — Starts text section

### Description

```
void cpdf_begin_text(int pdf document);
```

The `cpdf_begin_text` function starts a text section. It must be ended with `cpdf_end_text`.

#### Example 1. Text output

```
<?php cpdf_begin_text($pdf);  
cpdf_set_font($pdf, 16, "Helvetica", 4);  
cpdf_text($pdf, 100, 100, "Some text");  
cpdf_end_text($pdf) ?>
```

See also `cpdf_end_text`.

## **cpdf\_end\_text**

### **Name**

`cpdf_end_text` — Starts text section

### **Description**

```
void cpdf_end_text(int pdf document);
```

The `cpdf_end_text` function ends a text section which was started with `cpdf_begin_text`.

#### **Example 1. Text output**

```
<?php cpdf_begin_text($pdf);  
cpdf_set_font($pdf, 16, "Helvetica", 4);  
cpdf_text($pdf, 100, 100, "Some text");  
cpdf_end_text($pdf) ?>
```

See also `cpdf_begin_text`.

## **cpdf\_show**

### **Name**

`cpdf_show` — Output text at current position

## Description

```
void cpdf_show(int pdf document, string text);
```

The `cpdf_show` function outputs the string in *text* at the current position.

See also `cpdf_text`, `cpdf_begin_text`, `cpdf_end_text`.

## cpdf\_show\_xy

### Name

`cpdf_show_xy` — Output text at position

## Description

```
void cpdf_show_xy(int pdf document, string text, double x-koor, double  
y-koor, int mode);
```

The `cpdf_show_xy` function outputs the string *text* at position with coordinates (*x-koor*, *y-koor*). The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

**Note:** The function `cpdf_show_xy` is identical to `cpdf_text` without the optional parameters.

See also `cpdf_text`.

## cpdf\_text

### Name

cpdf\_text — Output text with parameters

### Description

```
void cpdf_text(int pdf document, string text, double x-koor, double y-koor,  
int mode, double orientation, int alignmode);
```

The `cpdf_text` function outputs the string *text* at position with coordinates (*x-koor*, *y-koor*). The optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit. The optional parameter *orientation* is the rotation of the text in degree. The optional parameter *alignmode* determines how the text is align. See the ClibPDF documentation for possible values.

See also `cpdf_show_xy`.

## cpdf\_set\_font

### Name

cpdf\_set\_font — Select the current font face and size

### Description

```
void cpdf_set_font(int pdf document, string font name, double size, int  
encoding);
```

The `cpdf_set_font` function sets the the current font face, font size and encoding. Currently only the standard postscript fonts are supported. The last parameter *encoding* can take the following values: 2 = macroman, 3 = macexpert, 4 = winansi. Any other value selects the font's buildin encoding.

## cpdf\_set\_leading

### Name

`cpdf_set_leading` — Sets distance between text lines

### Description

```
void cpdf_set_leading(int pdf document, double distance);
```

The `cpdf_set_leading` function sets the distance between text lines. This will be used if text is output by `cpdf_continue_text`.

See also `cpdf_continue_text`.

## cpdf\_set\_text\_rendering

### Name

`cpdf_set_text_rendering` — Determines how text is rendered

### Description

```
void cpdf_set_text_rendering(int pdf document, int mode);
```

The `cpdf_set_text_rendering` function determines how text is rendered. The possible values for *mode* are 0=fill text, 1=stroke text, 2=fill and stroke text, 3=invisible, 4=fill text and add it to clipping path, 5=stroke text and add it to clipping path, 6=fill and stroke text and add it to clipping path, 7=add it to clipping path.

## cpdf\_set\_horiz\_scaling

### Name

`cpdf_set_horiz_scaling` — Sets horizontal scaling of text

### Description

```
void cpdf_set_horiz_scaling(int pdf document, double scale);
```

The `cpdf_set_horiz_scaling` function sets the horizontal scaling to *scale* percent.

## cpdf\_set\_text\_rise

### Name

`cpdf_set_text_rise` — Sets the text rise

### Description

```
void cpdf_set_text_rise(int pdf document, double value);
```

The `cpdf_set_text_rise` function sets the text rising to *value* units.

## cpdf\_set\_text\_matrix

### Name

`cpdf_set_text_matrix` — Sets the text matrix

### Description

```
void cpdf_set_text_matrix(int pdf document, array matrix);
```

The `cpdf_set_text_matrix` function sets a matrix which describes a transformation applied on the current text font.

## cpdf\_set\_text\_pos

### Name

`cpdf_set_text_pos` — Sets text position

### Description

```
void cpdf_set_text_pos(int pdf document, double x-koor, double y-koor, int mode);
```

The `cpdf_set_text_pos` function sets the position of text for the next `cpdf_show` function call.

The last optional parameter *mode* determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_show`, `cpdf_text`.

## cpdf\_set\_char\_spacing

### Name

`cpdf_set_char_spacing` — Sets character spacing

### Description

```
void cpdf_set_char_spacing (int pdf document, double space);
```

The `cpdf_set_char_spacing` function sets the spacing between characters.

See also `cpdf_set_word_spacing`, `cpdf_set_leading`.

## cpdf\_set\_word\_spacing

### Name

`cpdf_set_word_spacing` — Sets spacing between words

### Description

```
void cpdf_set_word_spacing (int pdf document, double space);
```

The `cpdf_set_word_spacing` function sets the spacing between words.

See also `cpdf_set_char_spacing`, `cpdf_set_leading`.

## cpdf\_continue\_text

### Name

cpdf\_continue\_text — Output text in next line

### Description

```
void cpdf_continue_text (int pdf document, string text);
```

The `cpdf_continue_text` function outputs the string in *text* in the next line.

See also `cpdf_show_xy`, `cpdf_text`, `cpdf_set_leading`, `cpdf_set_text_pos`.

## cpdf\_stringwidth

### Name

cpdf\_stringwidth — Returns width of text in current font

### Description

```
double cpdf_stringwidth (int pdf document, string text);
```

The `cpdf_stringwidth` function returns the width of the string in *text*. It requires a font to be set before.

See also `cpdf_set_font`.

## cpdf\_save

### Name

`cpdf_save` — Saves current environment

### Description

```
void cpdf_save(int pdf document);
```

The `cpdf_save` function saves the current environment. It works like the postscript command `gsave`. Very useful if you want to translate or rotate an object without effecting other objects.

See also `cpdf_restore`.

## cpdf\_restore

### Name

`cpdf_restore` — Restores formerly saved environment

### Description

```
void cpdf_restore(int pdf document);
```

The `cpdf_restore` function restores the environment saved with `cpdf_save`. It works like the postscript command `grestore`. Very useful if you want to translate or rotate an object without effecting other objects.

#### Example 1. Save/Restore

```
<?php cpdf_save($pdf);  
// do all kinds of rotations, transformations, ...  
cpdf_restore($pdf) ?>
```

See also `cpdf_save`.

## **cpdf\_translate**

### **Name**

`cpdf_translate` — Sets origin of coordinate system

### **Description**

```
void cpdf_translate(int pdf document, double x-koor, double y-koor, int  
mode);
```

The `cpdf_translate` function set the origin of coordinate system to the point (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

## **cpdf\_scale**

### **Name**

`cpdf_scale` — Sets scaling

### **Description**

```
void cpdf_scale(int pdf document, double x-scale, double y-scale);
```

The `cpdf_scale` function set the scaling factor in both directions.

## cpdf\_rotate

### Name

`cpdf_rotate` — Sets rotation

### Description

```
void cpdf_rotate(int pdf document, double angle);
```

The `cpdf_rotate` function set the rotation in degrees to *angle*.

## cpdf\_setflat

### Name

`cpdf_setflat` — Sets flatness

### Description

```
void cpdf_setflat(int pdf document, double value);
```

The `cpdf_setflat` function set the flatness to a value between 0 and 100.

## cpdf\_setlinejoin

### Name

cpdf\_setlinejoin — Sets linejoin parameter

### Description

```
void cpdf_setlinejoin(int pdf document, long value);
```

The `cpdf_setlinejoin` function set the linejoin parameter between a value of 0 and 2. 0 = miter, 1 = round, 2 = bevel.

## cpdf\_setlinecap

### Name

cpdf\_setlinecap — Sets linecap aparameter

### Description

```
void cpdf_setlinecap(int pdf document, int value);
```

The `cpdf_setlinecap` function set the linecap parameter between a value of 0 and 2. 0 = butt end, 1 = round, 2 = projecting square.

## cpdf\_setmiterlimit

### Name

cpdf\_setmiterlimit — Sets miter limit

### Description

```
void cpdf_setmiterlimit(int pdf document, double value);
```

The `cpdf_setmiterlimit` function set the miter limit to a value greater or equal than 1.

## cpdf\_setlinewidth

### Name

cpdf\_setlinewidth — Sets line width

### Description

```
void cpdf_setlinewidth(int pdf document, double width);
```

The `cpdf_setlinewidth` function set the line width to *width*.

## cpdf\_setdash

### Name

cpdf\_setdash — Sets dash pattern

### Description

```
void cpdf_setdash(int pdf document, double white, double black);
```

The `cpdf_setdash` function set the dash pattern *white* white units and *black* black units. If both are 0 a solid line is set.

## cpdf\_moveto

### Name

cpdf\_moveto — Sets current point

### Description

```
void cpdf_moveto(int pdf document, double x-koor, double y-koor, int mode);
```

The `cpdf_moveto` function set the current point to the coordinates *x-koor* and *y-koor*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

## cpdf\_rmoveto

### Name

`cpdf_rmoveto` — Sets current point

### Description

```
void cpdf_rmoveto(int pdf document, double x-koor, double y-koor, int mode);
```

The `cpdf_rmoveto` function set the current point relative to the coordinates *x-koor* and *y-koor*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_moveto`.

## cpdf\_curveto

### Name

`cpdf_curveto` — Draws a curve

### Description

```
void cpdf_curveto(int pdf document, double x1, double y1, double x2, double y2, double x3, double y3, int mode);
```

The `cpdf_curveto` function draws a Bezier curve from the current point to the point (*x3*, *y3*) using (*x1*, *y1*) and (*x2*, *y2*) as control points.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_moveto`, `cpdf_rmoveto`, `cpdf_rlineto`, `cpdf_lineto`.

## cpdf\_lineto

### Name

cpdf\_lineto — Draws a line

### Description

```
void cpdf_lineto(int pdf document, double x-koor, double y-koor, int mode);
```

The `cpdf_lineto` function draws a line from the current point to the point with coordinates (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_moveto`, `cpdf_rmoveto`, `cpdf_curveto`.

## cpdf\_rlineto

### Name

cpdf\_rlineto — Draws a line

### Description

```
void cpdf_rlineto(int pdf document, double x-koor, double y-koor, int mode);
```

The `cpdf_rlineto` function draws a line from the current point to the relative point with coordinates (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_moveto`, `cpdf_rmoveto`, `cpdf_curveto`.

## cpdf\_circle

### Name

cpdf\_circle — Draw a circle

### Description

```
void cpdf_circle(int pdf document, double x-koor, double y-koor, double radius, int mode);
```

The `cpdf_circle` function draws a circle with center at point (*x-koor*, *y-koor*) and radius *radius*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_arc`.

## cpdf\_arc

### Name

cpdf\_arc — Draws an arc

### Description

```
void cpdf_arc(int pdf document, double x-koor, double y-koor, double radius, double start, double end, int mode);
```

The `cpdf_arc` function draws an arc with center at point (*x-koor*, *y-koor*) and radius *radius*, starting at angle *start* and ending at angle *end*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_circle`.

## **cpdf\_rect**

### **Name**

`cpdf_rect` — Draw a rectangle

### **Description**

```
void cpdf_rect(int pdf document, double x-koor, double y-koor, double width,  
double height, int mode);
```

The `cpdf_rect` function draws a rectangle with its lower left corner at point (*x-koor*, *y-koor*). This width is set to *width*. This height is set to *height*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

## **cpdf\_closepath**

### **Name**

`cpdf_closepath` — Close path

### **Description**

```
void cpdf_closepath(int pdf document);
```

The `cpdf_closepath` function closes the current path.

## cpdf\_stroke

### Name

cpdf\_stroke — Draw line along path

### Description

```
void cpdf_stroke(int pdf document);
```

The cpdf\_stroke function draws a line along current path.

See also cpdf\_closepath, cpdf\_closepath\_stroke.

## cpdf\_closepath\_stroke

### Name

cpdf\_closepath\_stroke — Close path and draw line along path

### Description

```
void cpdf_closepath_stroke(int pdf document);
```

The cpdf\_closepath\_stroke function is a combination of cpdf\_closepath and cpdf\_stroke. Than clears the path.

See also cpdf\_closepath, cpdf\_stroke.

## cpdf\_fill

### Name

cpdf\_fill — Fill current path

### Description

```
void cpdf_fill(int pdf document);
```

The `cpdf_fill` function fills the interior of the current path with the current fill color.

See also `cpdf_closepath`, `cpdf_stroke`, `cpdf_setgray_fill`, `cpdf_setgray`, `cpdf_setrgbcolor_fill`, `cpdf_setrgbcolor`.

## cpdf\_fill\_stroke

### Name

cpdf\_fill\_stroke — Fill and stroke current path

### Description

```
void cpdf_fill_stroke(int pdf document);
```

The `cpdf_fill_stroke` function fills the interior of the current path with the current fill color and draws current path.

See also `cpdf_closepath`, `cpdf_stroke`, `cpdf_fill`, `cpdf_setgray_fill`, `cpdf_setgray`, `cpdf_setrgbcolor_fill`, `cpdf_setrgbcolor`.

## cpdf\_closepath\_fill\_stroke

### Name

cpdf\_closepath\_fill\_stroke — Close, fill and stroke current path

### Description

```
void cpdf_closepath_fill_stroke (int pdf document);
```

The `cpdf_closepath_fill_stroke` function closes, fills the interior of the current path with the current fill color and draws current path.

See also `cpdf_closepath`, `cpdf_stroke`, `cpdf_fill`, `cpdf_setgray_fill`, `cpdf_setgray`, `cpdf_setrgbcolor_fill`, `cpdf_setrgbcolor`.

## cpdf\_clip

### Name

cpdf\_clip — Clips to current path

### Description

```
void cpdf_clip (int pdf document);
```

The `cpdf_clip` function clips all drawing to the current path.

## cpdf\_setgray\_fill

### Name

`cpdf_setgray_fill` — Sets filling color to gray value

### Description

```
void cpdf_setgray_fill(int pdf document, double value);
```

The `cpdf_setgray_fill` function sets the current gray value to fill a path.

See also `cpdf_setrgbcolor_fill`.

## cpdf\_setgray\_stroke

### Name

`cpdf_setgray_stroke` — Sets drawing color to gray value

### Description

```
void cpdf_setgray_stroke(int pdf document, double gray value);
```

The `cpdf_setgray_stroke` function sets the current drawing color to the given gray value.

See also `cpdf_setrgbcolor_stroke`.

## cpdf\_setgray

### Name

`cpdf_setgray` — Sets drawing and filling color to gray value

### Description

```
void cpdf_setgray(int pdf document, double gray value);
```

The `cpdf_setgray_stroke` function sets the current drawing and filling color to the given gray value.

See also `cpdf_setrgbcolor_stroke`, `cpdf_setrgbcolor_fill`.

## cpdf\_setrgbcolor\_fill

### Name

`cpdf_setrgbcolor_fill` — Sets filling color to rgb color value

### Description

```
void cpdf_setrgbcolor_fill(int pdf document, double red value, double green value, double blue value);
```

The `cpdf_setrgbcolor_fill` function sets the current rgb color value to fill a path.

See also `cpdf_setrgbcolor_stroke`, `cpdf_setrgbcolor`.

## cpdf\_setrgbcolor\_stroke

### Name

cpdf\_setrgbcolor\_stroke — Sets drawing color to rgb color value

### Description

```
void cpdf_setrgbcolor_stroke(int pdf document, double red value, double green value, double blue value);
```

The cpdf\_setrgbcolor\_stroke function sets the current drawing color to the given rgb color value.

See also cpdf\_setrgbcolor\_fill, cpdf\_setrgbcolor.

## cpdf\_setrgbcolor

### Name

cpdf\_setrgbcolor — Sets drawing and filling color to rgb color value

### Description

```
void cpdf_setrgbcolor(int pdf document, double red value, double green value, double blue value);
```

The cpdf\_setrgbcolor\_stroke function sets the current drawing and filling color to the given rgb color value.

See also cpdf\_setrgbcolor\_stroke, cpdf\_setrgbcolor\_fill.

## cpdf\_add\_outline

### Name

`cpdf_add_outline` — Adds bookmark for current page

### Description

```
void cpdf_add_outline(int pdf document, string text);
```

The `cpdf_add_outline` function adds a bookmark with text *text* that points to the current page.

#### Example 1. Adding a page outline

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
// ...
// some drawing
// ...
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

## cpdf\_set\_page\_animation

### Name

`cpdf_set_page_animation` — Sets duration between pages

## Description

```
void cpdf_set_page_animation(int pdf document, int transition, double duration);
```

The `cpdf_set_page_animation` function set the transition between following pages.

The value of *transition* can be

- 0 for none,
- 1 for two lines sweeping across the screen reveal the page,
- 2 for multiple lines sweeping across the screen reveal the page,
- 3 for a box reveals the page,
- 4 for a single line sweeping across the screen reveals the page,
- 5 for the old page dissolves to reveal the page,
- 6 for the dissolve effect moves from one screen edge to another,
- 7 for the old page is simply replaced by the new page (default)

The value of *duration* is the number of seconds between page flipping.

## cpdf\_import\_jpeg

### Name

`cpdf_import_jpeg` — Opens a JPEG image

### Description

```
int cpdf_open_jpeg(int pdf document, string file name, double x-koor, double y-koor, double angle, double width, double height, double x-scale, double y-scale, int mode);
```

The `cpdf_import_jpeg` function opens an image stored in the file with the name *file name*. The format of the image has to be jpeg. The image is placed on the current page at position (*x-koor*,

*y-koor*). The image is rotated by *angle* degrees.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_place_inline_image`,

## cpdf\_place\_inline\_image

### Name

`cpdf_place_inline_image` — Places an image on the page

### Description

```
void cpdf_place_inline_image(int pdf document, int image, double x-koor,
double y-koor, double angle, double width, double height, int mode);
```

The `cpdf_place_inline_image` function places an image created with the php image functions on the page at postion (*x-koor*, *y-koor*). The image can be scaled at the same time.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also `cpdf_import_jpeg`,

## cpdf\_add\_annotation

### Name

`cpdf_add_annotation` — Adds annotation

## Description

```
void cpdf_add_annotation(int pdf document, double llx, double lly, double  
urx, double ury, string title, string content, int mode);
```

The `cpdf_add_annotation` adds a note with the lower left corner at (*llx*, *lly*) and the upper right corner at (*urx*, *ury*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

## **VIII. Date and Time functions**

# checkdate

## Name

checkdate — validate a date/time

## Description

```
int checkdate(int month, int day, int year);
```

Returns true if the date given is valid; otherwise returns false. Checks the validity of the date formed by the arguments. A date is considered valid if:

- year is between 0 and 32767 inclusive
- month is between 1 and 12 inclusive
- day is within the allowed number of days for the given month. Leap years are taken into consideration.

# date

## Name

date — format a local time/date

## Description

```
string date(string format, int [timestamp] );
```

Returns a string formatted according to the given format string using the given *timestamp* or the current local time if no timestamp is given.

The following characters are recognized in the format string:

- a - "am" or "pm"
- A - "AM" or "PM"
- d - day of the month, 2 digits with leading zeros; i.e. "01" to "31"
- D - day of the week, textual, 3 letters; i.e. "Fri"
- F - month, textual, long; i.e. "January"
- h - hour, 12-hour format; i.e. "01" to "12"
- H - hour, 24-hour format; i.e. "00" to "23"
- g - hour, 12-hour format without leading zeros; i.e. "1" to "12"
- G - hour, 24-hour format without leading zeros; i.e. "0" to "23"
- i - minutes; i.e. "00" to "59"
- j - day of the month without leading zeros; i.e. "1" to "31"
- l (lowercase 'L') - day of the week, textual, long; i.e. "Friday"
- L - boolean for whether it is a leap year; i.e. "0" or "1"
- m - month; i.e. "01" to "12"
- n - month without leading zeros; i.e. "1" to "12"
- M - month, textual, 3 letters; i.e. "Jan"
- s - seconds; i.e. "00" to "59"
- S - English ordinal suffix, textual, 2 characters; i.e. "th", "nd"
- t - number of days in the given month; i.e. "28" to "31"
- U - seconds since the epoch
- w - day of the week, numeric, i.e. "0" (Sunday) to "6" (Saturday)
- Y - year, 4 digits; i.e. "1999"
- y - year, 2 digits; i.e. "99"
- z - day of the year; i.e. "0" to "365"
- Z - timezone offset in seconds (i.e. "-43200" to "43200")

Unrecognized characters in the format string will be printed as-is. The "Z" format will always return "0" when using `gmdate()`.

#### **Example 1. date example**

```
print (date("l dS of F Y h:i:s A"));
```

```
print ("July 1, 2000 is on a " . date("l", mktime(0,0,0,7,1,2000)));
```

It is possible to use `date` and `mktime` together to find dates in the future or the past.

#### Example 2. `date` and `mktime` example

```
$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
$lastmonth = mktime(0,0,0,date("m")-1,date("d"),date("Y"));
$nextyear = mktime(0,0,0,date("m"),date("d"),date("Y")+1);
```

To format dates in other languages, you should use the `setlocale` and `strftime` functions.

See also `gmdate` and `mktime`.

## getdate

### Name

`getdate` — get date/time information

### Description

```
array getdate(int timestamp);
```

Returns an associative array containing the date information of the timestamp as the following array elements:

- "seconds" - seconds
- "minutes" - minutes
- "hours" - hours
- "mday" - day of the month
- "wday" - day of the week, numeric

- "mon" - month, numeric
- "year" - year, numeric
- "yday" - day of the year, numeric; i.e. "299"
- "weekday" - day of the week, textual, full; i.e. "Friday"
- "month" - month, textual, full; i.e. "January"

## gettimeofday

### Name

gettimeofday — get current time

### Description

```
array gettimeofday(void);
```

This is an interface to gettimeofday(2). It returns an associative array containing the data returned from the system call.

- "sec" - seconds
- "usec" - microseconds
- "minuteswest" - minutes west of Greenwich
- "dsttime" - type of dst correction

# gmdate

## Name

gmdate — format a GMT/CUT date/time

## Description

```
string gmdate(string format, int timestamp);
```

Identical to the `date` function except that the time returned is Greenwich Mean Time (GMT). For example, when run in Finland (GMT +0200), the first line below prints "Jan 01 1998 00:00:00", while the second prints "Dec 31 1997 22:00:00".

### Example 1. gmdate example

```
echo date( "M d Y H:i:s",mktime(0,0,0,1,1,1998) );
echo gmdate( "M d Y H:i:s",mktime(0,0,0,1,1,1998) );
```

See also `date`, `mktime` and `gmmktime`.

# gmmktime

## Name

gmmktime — get UNIX timestamp for a GMT date

## Description

```
int gmmktime(int hour, int minute, int second, int month, int day, int year,
int [is_dst]);
```

Identical to `mktime` except the passed parameters represents a GMT date.

## gmstrftime

### Name

`gmstrftime` — format a GMT/CUT time/date according to locale settings

### Description

```
string gmstrftime(string format, int timestamp);
```

Behaves the same as `strftime` except that the time returned is Greenwich Mean Time (GMT). For example, when run in Eastern Standard Time (GMT -0500), the first line below prints "Dec 31 1998 20:00:00", while the second prints "Jan 01 1999 01:00:00".

#### Example 1. `gmstrftime` example

```
setlocale ('LC_TIME', 'en_US');
echo strftime ("%b %d %Y %H:%M:%S", mktime(20,0,0,12,31,98))."\n";
echo gmstrftime ("%b %d %Y %H:%M:%S", mktime(20,0,0,12,31,98))."\n";
```

See also `strftime`.

## microtime

### Name

`microtime` — return current UNIX timestamp with microseconds

## Description

```
string microtime(void);
```

Returns the string "msec sec" where sec is the current time measured in the number of seconds since the Unix Epoch (0:00:00 January 1, 1970 GMT), and msec is the microseconds part. This function is only available on operating systems that support the `gettimeofday()` system call.

See also `time`.

## mktime

### Name

`mktime` — get UNIX timestamp for a date

### Description

```
int mktime(int hour, int minute, int second, int month, int day, int year,  
int [is_dst]);
```

*Warning:* Note the strange order of arguments, which differs from the order of arguments in a regular UNIX `mktime()` call and which does not lend itself well to leaving out parameters from right to left (see below). It is a common error to mix these values up in a script.

Returns the Unix timestamp corresponding to the arguments given. This timestamp is a long integer containing the number of seconds between the Unix Epoch (January 1 1970) and the time specified.

Arguments may be left out in order from right to left; any arguments thus omitted will be set to the current value according to the local date and time.

`is_dst` can be set to 1 if the time is during daylight savings time, 0 if it is not, or -1 (the default) if it is unknown whether the time is within daylight savings time or not.

**Note:** `is_dst` was added in 3.0.10.

`mktime` is useful for doing date arithmetic and validation, as it will automatically calculate the correct value for out-of-range input. For example, each of the following lines produces the string "Jan-01-1998".

#### Example 1. `mktime` example

```
echo date( "M-d-Y", mktime(0,0,0,12,32,1997) );
echo date( "M-d-Y", mktime(0,0,0,13,1,1997) );
echo date( "M-d-Y", mktime(0,0,0,1,1,1998) );
```

See also `date` and `time`.

## strftime

### Name

`strftime` — format a local time/date according to locale settings

### Description

```
string strftime(string format, int timestamp);
```

Returns a string formatted according to the given format string using the given *timestamp* or the current local time if no timestamp is given. Month and weekday names and other language dependent strings respect the current locale set with `setlocale`.

The following conversion specifiers are recognized in the format string:

- `%a` - abbreviated weekday name according to the current locale
- `%A` - full weekday name according to the current locale
- `%b` - abbreviated month name according to the current locale
- `%B` - full month name according to the current locale
- `%c` - preferred date and time representation for the current locale

- %d - day of the month as a decimal number (range 00 to 31)
- %H - hour as a decimal number using a 24-hour clock (range 00 to 23)
- %I - hour as a decimal number using a 12-hour clock (range 01 to 12)
- %j - day of the year as a decimal number (range 001 to 366)
- %m - month as a decimal number (range 1 to 12)
- %M - minute as a decimal number
- %p - either 'am' or 'pm' according to the given time value, or the corresponding strings for the current locale
- %S - second as a decimal number
- %U - week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week
- %W - week number of the current year as a decimal number, starting with the first Monday as the first day of the first week
- %w - day of the week as a decimal, Sunday being 0
- %x - preferred date representation for the current locale without the time
- %X - preferred time representation for the current locale without the date
- %y - year as a decimal number without a century (range 00 to 99)
- %Y - year as a decimal number including the century
- %Z - time zone or name or abbreviation
- %% - a literal '%' character

### Example 1. strftime example

```
setlocale ("LC_TIME", "C");
print(strftime("%A in Finnish is "));
setlocale ("LC_TIME", "fi_FI");
print(strftime("%A, in French "));
setlocale ("LC_TIME", "fr_CA");
print(strftime("%A and in German "));
setlocale ("LC_TIME", "de_DE");
print(strftime("%A.\n"));
```

This example works if you have the respective locales installed in your system.

See also `setlocale` and `mktime`.

# time

## Name

`time` — return current UNIX timestamp

## Description

```
int time(void);
```

Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

See also `date`.

# IX. Database (dbm-style)

## abstraction layer functions

These functions build the foundation for accessing Berkeley DB style databases.

This is a general abstraction layer for several file-based databases. As such, functionality is limited to a subset of features modern databases such as Sleepycat Software's DB2 () support. (This is not to be confused with IBM's DB2 software, which is supported through the ODBC functions.)

The behaviour of various aspects depend on the implementation of the underlying database. Functions such as `dba_optimize` and `dba_sync` will do what they promise for one database and will do nothing for others.

The following handlers are supported:

- `dbm` is the oldest (original) type of Berkeley DB style databases. You should avoid it, if possible. We do not support the compatibility functions built into DB2 and `gdbm`, because they are only compatible on the source code level, but cannot handle the original `dbm` format.
- `ndbm` is a newer type and more flexible than `dbm`. It still has most of the arbitrary limits of `dbm` (therefore it is deprecated).
- `gdbm` is the GNU database manager ().
- `db2` is Sleepycat Software's DB2 (). It is described as "a programmatic toolkit that provides high-performance built-in database support for both standalone and client/server applications."
- `cdb` is "a fast, reliable, lightweight package for creating and reading constant databases." It is from the author of `qmail` and can be found here (). Since it is constant, we support only reading operations.

### Example 1. DBA example

```
<?php

$id = dba_open("/tmp/test.db", "n", "db2");

if(!$id) {
    echo "dba_open failed\n";
    exit;
}

dba_replace("key", "This is an example!", $id);

if(dba_exists("key", $id)) {
```

```

        echo dba_fetch("key", $id);
        dba_delete("key", $id);
    }

    dba_close($id);
?>

```

DBA is binary safe and does not have any arbitrary limits. It inherits all limits set by the underlying database implementation.

All file-based databases must provide a way of setting the file mode of a new created database, if that is possible at all. The file mode is commonly passed as the fourth argument to `dba_open` or `dba_popen`.

You can access all entries of a database in a linear way by using the `dba_firstkey` and `dba_nextkey` functions. You may not change the database while traversing it.

### Example 2. Traversing a database

```

<?php

# ...open database...

$key = dba_firstkey($id);

while($key != false) {
    if(...) { # remember the key to perform some action later
        $handle_later[] = $key;
    }
    $key = dba_nextkey($id);
}

for($i = 0; $i < count($handle_later); $i++)
    dba_delete($handle_later[$i], $id);

?>

```

## dba\_close

### Name

dba\_close — Close database

### Description

```
void dba_close(int handle);
```

dba\_close closes the established database and frees all resources specified by *handle*.

*handle* is a database handle returned by dba\_open.

dba\_close does not return any value.

See also: dba\_open dba\_popen

## dba\_delete

### Name

dba\_delete — Delete entry specified by key

### Description

```
string dba_delete(string key, int handle);
```

dba\_delete deletes the entry specified by *key* from the database specified with *handle*.

*key* is the key of the entry which is deleted.

*handle* is a database handle returned by dba\_open.

dba\_delete returns true or false, if the entry is deleted or not deleted, respectively.

See also: `dba_exists` `dba_fetch` `dba_insert` `dba_replace`

## dba\_exists

### Name

`dba_exists` — Check whether key exists

### Description

```
bool dba_exists(string key, int handle);
```

`dba_exists` checks whether the specified *key* exists in the database specified by *handle*.

*key* is the key the check is performed for.

*handle* is a database handle returned by `dba_open`.

`dba_exists` returns true or false, if the key is found or not found, respectively.

See also: `dba_fetch` `dba_delete` `dba_insert` `dba_replace`

## dba\_fetch

### Name

`dba_fetch` — Fetch data specified by key

### Description

```
string dba_fetch(string key, int handle);
```

`dba_fetch` fetches the data specified by *key* from the database specified with *handle*.

*key* is the key the data is specified by.

*handle* is a database handle returned by `dba_open`.

`dba_fetch` returns the associated string or false, if the key/data pair is found or not found, respectively.

See also: `dba_exists` `dba_delete` `dba_insert` `dba_replace`

## dba\_firstkey

### Name

`dba_firstkey` — Fetch first key

### Description

```
string dba_firstkey(int handle);
```

`dba_firstkey` returns the first key of the database specified by *handle* and resets the internal key pointer. This permits a linear search through the whole database.

*handle* is a database handle returned by `dba_open`.

`dba_firstkey` returns the key or false depending on whether it succeeds or fails, respectively.

See also: `dba_nextkey`

## dba\_insert

### Name

`dba_insert` — Insert entry

## Description

```
bool dba_insert (string key, string value, int handle);
```

`dba_insert` inserts the entry described with *key* and *value* into the database specified by *handle*. It fails, if an entry with the same *key* already exists.

*key* is the key of the entry to be inserted.

*value* is the value to be inserted.

*handle* is a database handle returned by `dba_open`.

`dba_insert` returns true or false, depending on whether it succeeds or fails, respectively.

See also: `dba_exists` `dba_delete` `dba_fetch` `dba_replace`

## dba\_nextkey

### Name

`dba_nextkey` — Fetch next key

## Description

```
string dba_nextkey (int handle);
```

`dba_nextkey` returns the next key of the database specified by *handle* and increments the internal key pointer.

*handle* is a database handle returned by `dba_open`.

`dba_nextkey` returns the key or false depending on whether it succeeds or fails, respectively.

See also: `dba_firstkey`

## dba\_popen

### Name

dba\_popen — Open database persistently

### Description

```
int dba_popen(string path, string mode, string handler, [...]);
```

dba\_popen establishes a persistent database instance for *path* with *mode* using *handler*.

*path* is commonly a regular path in your filesystem.

*mode* is "r" for read access, "w" for read/write access to an already existing database, "c" for read/write access and database creation if it doesn't currently exist, and "n" for create, truncate and read/write access.

*handler* is the name of the handler which shall be used for accessing *path*. It is passed all optional parameters given to dba\_popen and can act on behalf of them.

dba\_popen returns a positive handler id or false, in the case the open is successful or fails, respectively.

See also: dba\_open dba\_close

## dba\_open

### Name

dba\_open — Open database

### Description

```
int dba_open(string path, string mode, string handler, [...]);
```

`dba_open` establishes a database instance for *path* with *mode* using *handler*.

*path* is commonly a regular path in your filesystem.

*mode* is "r" for read access, "w" for read/write access to an already existing database, "c" for read/write access and database creation if it doesn't currently exist, and "n" for create, truncate and read/write access.

*handler* is the name of the handler which shall be used for accessing *path*. It is passed all optional parameters given to `dba_open` and can act on behalf of them.

`dba_open` returns a positive handler id or false, in the case the open is successful or fails, respectively.

See also: `dba_popen` `dba_close`

## dba\_optimize

### Name

`dba_optimize` — Optimize database

### Description

```
bool dba_optimize(int handle);
```

`dba_optimize` optimizes the underlying database specified by *handle*.

*handle* is a database handle returned by `dba_open`.

`dba_optimize` returns true or false, if the optimization succeeds or fails, respectively.

See also: `dba_sync`

## dba\_replace

### Name

dba\_replace — Replace or insert entry

### Description

```
bool dba_replace(string key, string value, int handle);
```

dba\_replace replaces or inserts the entry described with *key* and *value* into the database specified by *handle*.

*key* is the key of the entry to be inserted.

*value* is the value to be inserted.

*handle* is a database handle returned by dba\_open.

dba\_replace returns true or false, depending on whether it succeeds or fails, respectively.

See also: dba\_exists dba\_delete dba\_fetch dba\_insert

## dba\_sync

### Name

dba\_sync — Synchronize database

### Description

```
bool dba_sync(int handle);
```

dba\_sync synchronizes the database specified by *handle*. This will probably trigger a physical write to disk, if supported.

*handle* is a database handle returned by `dba_open`.

`dba_sync` returns true or false, if the synchronization succeeds or fails, respectively.

See also: `dba_optimize`

## X. dBase functions

These functions allow you to access records stored in dBase-format (dbf) databases.

There is no support for indexes or memo fields. There is no support for locking, too. Two concurrent webserver processes modifying the same dBase file will very likely ruin your database.

Unlike SQL databases, dBase "databases" cannot change the database definition afterwards. Once the file is created, the database definition is fixed. There are no indexes that speed searching or otherwise organize your data. dBase files are simple sequential files of fixed length records. Records are appended to the end of the file and delete records are kept until you call `dbase_pack()`.

We recommend that you do not use dBase files as your production database. Choose any real SQL server instead; MySQL or Postgres are common choices with PHP. dBase support is here to allow you to import and export data to and from your web database, since the file format is commonly understood with Windows spreadsheets and organizers. Import and export of data is about all that dBase support is good for.

# dbase\_create

## Name

dbase\_create — creates a dBase database

## Description

```
int dbase_create(string filename, array fields);
```

The *fields* parameter is an array of arrays, each array describing the format of one field in the database. Each field consists of a name, a character indicating the field type, a length, and a precision.

The types of fields available are:

L

Boolean. These do not have a length or precision.

M

Memo. (Note that these aren't supported by PHP.) These do not have a length or precision.

D

Date (stored as YYYYMMDD). These do not have a length or precision.

N

Number. These have both a length and a precision (the number of digits after the decimal point).

C

String.

If the database is successfully created, a `dbase_identifier` is returned, otherwise `false` is returned.

### Example 1. Creating a dBase database file

```
// "database" name
```

```

$dbname = "/tmp/test.dbf";

// database "definition"
$def =
    array(
        array("date",      "D"),
        array("name",      "C",  50),
        array("age",       "N",   3, 0),
        array("email",     "C", 128),
        array("ismember",  "L")
    );

// creation
if (!dbase_create($dbname, $def))
    print "<strong>Error!</strong>";

```

## dbase\_open

### Name

dbase\_open — opens a dBase database

### Description

```
int dbase_open(string filename, int flags);
```

The flags correspond to those for the open() system call. (Typically 0 means read-only, 1 means write-only, and 2 means read and write.)

Returns a dbase\_identifier for the opened database, or false if the database couldn't be opened.

## **dbase\_close**

### **Name**

`dbase_close` — close a dBase database

### **Description**

```
bool dbase_close(int dbase_identifier);
```

Closes the database associated with *dbase\_identifier*.

## **dbase\_pack**

### **Name**

`dbase_pack` — packs a dBase database

### **Description**

```
bool dbase_pack(int dbase_identifier);
```

Packs the specified database (permanently deleting all records marked for deletion using `dbase_delete_record`).

## dbase\_add\_record

### Name

`dbase_add_record` — add a record to a dBase database

### Description

```
bool dbase_add_record(int dbase_identifier, array record);
```

Adds the data in the *record* to the database. If the number of items in the supplied record isn't equal to the number of fields in the database, the operation will fail and false will be returned.

## dbase\_replace\_record

### Name

`dbase_replace_record` — replace a record in a dBase database

### Description

```
bool dbase_replace_record(int dbase_identifier, array record, int  
dbase_record_number);
```

Replaces the data associated with the record *record\_number* with the data in the *record* in the database. If the number of items in the supplied record is not equal to the number of fields in the database, the operation will fail and false will be returned.

*dbase\_record\_number* is an integer which spans from 1 to the number of records in the database (as returned by `dbase_numrecords`).

## dbase\_delete\_record

### Name

`dbase_delete_record` — deletes a record from a dBase database

### Description

```
bool dbase_delete_record(int dbase_identifier, int record);
```

Marks *record* to be deleted from the database. To actually remove the record from the database, you must also call `dbase_pack`.

## dbase\_get\_record

### Name

`dbase_get_record` — gets a record from a dBase database

### Description

```
array dbase_get_record(int dbase_identifier, int record);
```

Returns the data from *record* in an array. The array is indexed starting at 0, and includes an associative member named 'deleted' which is set to 1 if the record has been marked for deletion (see `dbase_delete_record`).

Each field is converted to the appropriate PHP type. (Dates are left as strings.)

## dbase\_get\_record\_with\_names

### Name

`dbase_get_record_with_names` — gets a record from a dBase database as an associative array

### Description

```
array dbase_get_record_with_names (int dbase_identifier, int record);
```

Returns the data from *record* in an associative array. The array also includes an associative member named 'deleted' which is set to 1 if the record has been marked for deletion (see `dbase_delete_record`).

Each field is converted to the appropriate PHP type. (Dates are left as strings.)

## dbase\_numfields

### Name

`dbase_numfields` — find out how many fields are in a dBase database

### Description

```
int dbase_numfields (int dbase_identifier);
```

Returns the number of fields (columns) in the specified database. Field numbers are between 0 and `dbase_numfields($db)-1`, while record numbers are between 1 and `dbase_numrecords($db)`.

#### Example 1. Using `dbase_numfields`

```
$rec = dbase_get_record($db, $recno);  
$nf  = dbase_numfields($db);  
for ($i=0; $i < $nf; $i++) {
```

```
        print $rec[$i]."<br>\n";  
    }
```

## **dbase\_numrecords**

### **Name**

`dbase_numrecords` — find out how many records are in a dBase database

### **Description**

```
int dbase_numrecords(int dbase_identifier);
```

Returns the number of records (rows) in the specified database. Record numbers are between 1 and `dbase_numrecords($db)`, while field numbers are between 0 and `dbase_numfields($db)-1`.

# XI. dbm functions

These functions allow you to store records stored in a dbm-style database. This type of database (supported by the Berkeley db, gdbm, and some system libraries, as well as a built-in flatfile library) stores key/value pairs (as opposed to the full-blown records supported by relational databases).

## Example 1. dbm example

```
$dbm = dbmopen("lastseen", "w");
if (dbmexists($dbm, $userid)) {
    $last_seen = dbmfetch($dbm, $userid);
} else {
    dbminsert($dbm, $userid, time());
}
do_stuff();
dbmreplace($dbm, $userid, time());
dbmclose($dbm);
```

# dbmopen

## Name

dbmopen — opens a dbm database

## Description

```
int dbmopen(string filename, string flags);
```

The first argument is the full-path filename of the dbm file to be opened and the second is the file open mode which is one of "r", "n", "c" or "w" for read-only, new (implies read-write, and most likely will truncate an already-existing database of the same name), create (implies read-write, and will not truncate an already-existing database of the same name) and read-write respectively.

Returns an identifier to be passed to the other dbm functions on success, or false on failure.

If ndbm support is used, ndbm will actually create filename.dir and filename.pag files. gdbm only uses one file, as does the internal flat-file support, and Berkeley db creates a filename.db file. Note that PHP does its own file locking in addition to any file locking that may be done by the dbm library itself. PHP does not delete the .lck files it creates. It uses these files simply as fixed inodes on which to do the file locking. For more information on dbm files, see your Unix man pages, or obtain GNU's gdbm from <ftp://prep.ai.mit.edu/pub/gnu>.

# dbmclose

## Name

dbmclose — closes a dbm database

## Description

```
bool dbmclose(int dbm_identifier);
```

Unlocks and closes the specified database.

## dbmexists

### Name

`dbmexists` — tells if a value exists for a key in a dbm database

### Description

```
bool dbmexists(int dbm_identifier, string key);
```

Returns true if there is a value associated with the *key*.

## dbmfetch

### Name

`dbmfetch` — fetches a value for a key from a dbm database

### Description

```
string dbmfetch(int dbm_identifier, string key);
```

Returns the value associated with *key*.

## dbminsert

### Name

dbminsert — inserts a value for a key in a dbm database

### Description

```
int dbminsert(int dbm_identifier, string key, string value);
```

Adds the value to the database with the specified key.

Returns -1 if the database was opened read-only, 0 if the insert was successful, and 1 if the specified key already exists. (To replace the value, use dbmreplace.)

## dbmreplace

### Name

dbmreplace — replaces the value for a key in a dbm database

### Description

```
bool dbmreplace(int dbm_identifier, string key, string value);
```

Replaces the value for the specified key in the database.

This will also add the key to the database if it didn't already exist.

## dbmdelete

### Name

`dbmdelete` — deletes the value for a key from a dbm database

### Description

```
bool dbmdelete(int dbm_identifier, string key);
```

Deletes the value for *key* in the database.

Returns false if the key didn't exist in the database.

## dbmfirstkey

### Name

`dbmfirstkey` — retrieves the first key from a dbm database

### Description

```
string dbmfirstkey(int dbm_identifier);
```

Returns the first key in the database. Note that no particular order is guaranteed since the database may be built using a hash-table, which doesn't guarantee any ordering.

## dbmnextkey

### Name

`dbmnextkey` — retrieves the next key from a dbm database

### Description

```
string dbmnextkey(int dbm_identifier, string key);
```

Returns the next key after *key*. By calling `dbmfirstkey` followed by successive calls to `dbmnextkey` it is possible to visit every key/value pair in the dbm database. For example:

**Example 1. Visiting every key/value pair in a dbm database.**

```
$key = dbmfirstkey($dbm_id);  
while ($key) {  
    echo "$key = " . dbmfetch($dbm_id, $key) . "\n";  
    $key = dbmnextkey($dbm_id, $key);  
}
```

## dblist

### Name

`dblist` — describes the dbm-compatible library being used

### Description

```
string dblist(void);
```

## **XII. Directory functions**

## chdir

### Name

`chdir` — change directory

### Description

```
int chdir(string directory);
```

Changes PHP's current directory to *directory*. Returns FALSE if unable to change directory, TRUE otherwise.

## dir

### Name

`dir` — directory class

### Description

```
new dir(string directory);
```

A pseudo-object oriented mechanism for reading a directory. The given *directory* is opened. Two properties are available once directory has been opened. The `handle` property can be used with other directory functions such as `readdir`, `rewinddir` and `closedir`. The `path` property is set to path the directory that was opened. Three methods are available: `read`, `rewind` and `close`.

#### Example 1. Dir() Example

```
$d = dir("/etc");  
echo "Handle: " . $d->handle . "<br>\n";  
echo "Path: " . $d->path . "<br>\n";
```

```
while($entry=$d->read()) {  
    echo $entry."<br>\n";  
}  
$d->close();
```

## closedir

### Name

`closedir` — close directory handle

### Description

```
void closedir(int dir_handle);
```

Closes the directory stream indicated by *dir\_handle*. The stream must have previously been opened by `opendir`.

## opendir

### Name

`opendir` — open directory handle

### Description

```
int opendir(string path);
```

Returns a directory handle to be used in subsequent `closedir`, `readdir`, and `rewinddir` calls.

# readdir

## Name

`readdir` — read entry from directory handle

## Description

```
string readdir(int dir_handle);
```

Returns the filename of the next file from the directory. The filenames are not returned in any particular order.

### Example 1. List all files in the current directory

```
<?php
    $handle=opendir('.');
    echo "Directory handle: $handle\n";
    echo "Files:\n";
    while ($file = readdir($handle)) {
        echo "$file\n";
    }
    closedir($handle);
?>
```

# rewinddir

## Name

`rewinddir` — rewind directory handle

## Description

```
void rewinddir(int dir_handle);
```

Resets the directory stream indicated by *dir\_handle* to the beginning of the directory.

## **XIII. Dynamic Loading functions**

# dl

## Name

dl — load a PHP extension at runtime

## Description

```
int dl(string library);
```

Loads the PHP extension defined in *library*. See also the `extension_dir` configuration directive.

## **XIV. Program Execution functions**

# escapeshellcmd

## Name

escapeshellcmd — escape shell metacharacters

## Description

```
string escapeshellcmd(string command);
```

EscapeShellCmd escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands. This function should be used to make sure that any data coming from user input is escaped before this data is passed to the `exec` or `system` functions. A standard use would be:

```
system(EscapeShellCmd($cmd))
```

# exec

## Name

exec — Execute an external program

## Description

```
string exec(string command, string [array], int [return_var]);
```

`exec` executes the given *command*, however it does not output anything. It simply returns the last line from the result of the command. If you need to execute a command and have all the data from the command passed directly back without any interference, use the `PassThru` function.

If the *array* argument is present, then the specified array will be filled with every line of output from the command. Note that if the array already contains some elements, `exec` will append to the end of the array. If you do not want the function to append elements, call `unset` on the array before passing it to `exec`.

If the *return\_var* argument is present along with the *array* argument, then the return status of the executed command will be written to this variable.

Note that if you are going to allow data coming from user input to be passed to this function, then you should be using `EscapeShellCmd` to make sure that users cannot trick the system into executing arbitrary commands.

See also `system`, `PassThru`, `popen` and `EscapeShellCmd`.

## system

### Name

`system` — Execute an external program and display output

### Description

```
string system(string command, int [return_var]);
```

`System` is just like the C version of the function in that it executes the given *command* and outputs the result. If a variable is provided as the second argument, then the return status code of the executed command will be written to this variable.

Note, that if you are going to allow data coming from user input to be passed to this function, then you should be using the `EscapeShellCmd` function to make sure that users cannot trick the system into executing arbitrary commands.

The `System` call also tries to automatically flush the web server's output buffer after each line of output if PHP is running as a server module.

If you need to execute a command and have all the data from the command passed directly back without any interference, use the `PassThru` function. See also the `exec` and `popen` functions.

# passthru

## Name

`passthru` — Execute an external program and display raw output

## Description

```
string passthru(string command, int [return_var]);
```

The `passthru` function is similar to the `Exec` function in that it executes a *command*. If the *return\_var* argument is present, the return status of the Unix command will be placed here. This function should be used in place of `Exec` or `System` when the output from the Unix command is binary data which needs to be passed directly back to the browser. A common use for this is to execute something like the `pbmplus` utilities that can output an image stream directly. By setting the content-type to *image/gif* and then calling a `pbmplus` program to output a gif, you can create PHP scripts that output images directly.

See also `exec` and `fpassthru`.

## XV. Forms Data Format functions

Forms Data Format (FDF) is a format for handling forms within PDF documents. You should read the documentation at <http://partners.adobe.com/asn/developer/acrosdk/forms.html> for more information on what FDF is and how it is used in general.

**Note:** Currently Adobe only provides a libc5 compatible version for Linux. Tests with glibc2 resulted in a segmentation fault. If somebody is able to make it work, please comment on this page.

**Note:** If you run into problems configuring php with fdfTk support, check whether the header file `FdfTk.h` and the library `libFdfTk.so` are at the right place. They should be in `fdftk-dir/include` and `fdftk-dir/lib`. This will not be the case if you just unpack the `FdfTk` distribution.

The general idea of FDF is similar to HTML forms. The difference is basically the format how filled in data is transmitted to the server when the submit button is pressed (this is actually the Form Data Format) and the format of the form itself (which is the Portable Document Format, PDF). Processing the FDF data is one of the features provided by the `fdf` functions. But there is more. One may as well take an existing PDF form and populated the input fields with data without modifying the form itself. In such a case one would create a FDF document (`fdf_create`) set the values of each input field (`fdf_set_value`) and associate it with a PDF form (`fdf_set_file`). Finally it has to be sent to the browser with MIME type `application/vnd.fdf`. The Acrobat reader plugin of your browser recognizes the MIME type, reads the associated PDF form and fills in the data from the FDF document.

The following examples shows just the evaluation of form data.

### Example 1. Evaluating a FDF document

```
<?php
// Save the FDF data into a temp file
$fdffp = fopen("test.fdf", "w");
fwrite($fdffp, $HTTP_FDF_DATA, strlen($HTTP_FDF_DATA));
fclose($fdffp);

// Open temp file and evaluate data
// The pdf form contained several input text fields with the names
// volume, date, comment, publisher, preparer, and two checkboxes
// show_publisher and show_preparer.
$fdf = fdf_open("test.fdf");
$volume = fdf_get_value($fdf, "volume");
echo "The volume field has the value '<B>$volume</B>'<BR>";
```

```
$date = fdf_get_value($fdf, "date");
echo "The date field has the value '<B>$date</B>'  
<BR>";

$comment = fdf_get_value($fdf, "comment");
echo "The comment field has the value '<B>$comment</B>'  
<BR>";

if(fdf_get_value($fdf, "show_publisher") == "On") {
    $publisher = fdf_get_value($fdf, "publisher");
    echo "The publisher field has the value '<B>$publisher</B>'  
<BR>";
} else
    echo "Publisher shall not be shown.<BR>";

if(fdf_get_value($fdf, "show_preparer") == "On") {
    $preparer = fdf_get_value($fdf, "preparer");
    echo "The preparer field has the value '<B>$preparer</B>'  
<BR>";
} else
    echo "Preparer shall not be shown.<BR>";
fdf_close($fdf);
?>
```

# fdf\_open

## Name

fdf\_open — Open a FDF document

## Description

```
int fdf_open(string filename);
```

The `fdf_open` function opens a file with form data. This file must contain the data as returned from a PDF form. Currently, the file has to be created 'manually' by using `fopen` and writing the content of `HTTP_FDF_DATA` with `fwrite` into it. A mechanism like for HTML form data where for each input field a variable is created does not exist.

### Example 1. Accessing the form data

```
<?php
// Save the FDF data into a temp file
$fdffp = fopen("test.fdf", "w");
fwrite($fdffp, $HTTP_FDF_DATA, strlen($HTTP_FDF_DATA));
fclose($fdffp);

// Open temp file and evaluate data
$fdf = fdf_open("test.fdf");
...
fdf_close($fdf);
?>
```

See also `fdf_close`.

## fdf\_close

### Name

`fdf_close` — Close an FDF document

### Description

```
void fdf_close(int fdf_document);
```

The `fdf_close` function closes the FDF document.

See also `fdf_open`.

## fdf\_create

### Name

`fdf_create` — Create a new FDF document

### Description

```
int fdf_create(void );
```

The `fdf_create` creates a new FDF document. This function is needed if one would like to populate input fields in a PDF document with data.

#### Example 1. Populating a PDF document

```
<?php
$outfdf = fdf_create();
fdf_set_value($outfdf, "volume", $volume, 0);
```

```
fdf_set_file($outfdf, "http://testfdf/resultlabel.pdf");
fdf_save($outfdf, "outtest.fdf");
fdf_close($outfdf);
Header("Content-type: application/vnd.fdf");
$fp = fopen("outtest.fdf", "r");
fpassthru($fp);
unlink("outtest.fdf");
?>
```

See also `fdf_close`, `fdf_save`, `fdf_open`.

## fdf\_save

### Name

`fdf_save` — Save a FDF document

### Description

```
int fdf_save(string filename);
```

The `fdf_save` function saves a FDF document. The FDF Toolkit provides a way to output the document to stdout if the parameter *filename* is `'.'`. This does not work if PHP is used as an apache module. In such a case one will have to write to a file and use e.g. `fpassthru` to output it.

See also `fdf_close` and example for `fdf_create`.

## fdf\_get\_value

### Name

`fdf_get_value` — Get the value of a field

### Description

```
string fdf_get_value(int fdf_document, string fieldname);
```

The `fdf_get_value` function returns the value of a field.

See also `fdf_set_value`.

## fdf\_set\_value

### Name

`fdf_set_value` — Set the value of a field

### Description

```
void fdf_set_value(int fdf_document, string fieldname, string value, int  
isName);
```

The `fdf_set_value` function sets the value of a field. The last parameter determines if the field value is to be converted to a PDF Name (`isName = 1`) or set to a PDF String (`isName = 0`).

See also `fdf_get_value`.

## fdf\_next\_field\_name

### Name

`fdf_next_field_name` — Get the next field name

### Description

```
string fdf_next_field_name(int fdf_document, string fieldname);
```

The `fdf_next_field_name` function returns the name of the field after the field in *fieldname* or the field name of the first field if the second paramter is NULL.

See also `fdf_set_field`, `fdf_get_field`.

## fdf\_set\_ap

### Name

`fdf_set_ap` — Set the appearance of a field

### Description

```
void fdf_set_ap(int fdf_document, string field_name, int face, string  
filename, int page_number);
```

The `fdf_set_ap` function sets the appearance of a field (i.e. the value of the /AP key). The possible values of *face* are 1=FDFNormalAP, 2=FDFRolloverAP, 3=FDFFDownAP.

## fdf\_set\_status

### Name

`fdf_set_status` — Set the value of the /STATUS key

### Description

```
void fdf_set_status(int fdf_document, string status);
```

The `fdf_set_status` sets the value of the /STATUS key.

See also `fdf_get_status`.

## fdf\_get\_status

### Name

`fdf_get_status` — Get the value of the /STATUS key

### Description

```
string fdf_get_status(int fdf_document);
```

The `fdf_get_status` returns the value of the /STATUS key.

See also `fdf_set_status`.

## fdf\_set\_file

### Name

`fdf_set_file` — Set the value of the /F key

### Description

```
void fdf_set_file(int fdf_document, string filename);
```

The `fdf_set_file` sets the value of the /F key. The /F key is just a reference to a PDF form which is to be populated with data. In a web environment it is a URL (e.g. <http://testfdf/resultlabel.pdf>).

See also `fdf_get_file` and example for `fdf_create`.

## fdf\_get\_file

### Name

`fdf_get_file` — Get the value of the /F key

### Description

```
string fdf_get_file(int fdf_document);
```

The `fdf_set_file` returns the value of the /F key.

See also `fdf_set_file`.

## **XVI. filePro functions**

These functions allow read-only access to data stored in filePro databases.

filePro is a registered trademark of Fiserv, Inc. You can find more information about filePro at <http://www.fileproplus.com/>.

## filepro

### Name

`filepro` — read and verify the map file

### Description

```
bool filepro(string directory);
```

This reads and verifies the map file, storing the field count and info.

No locking is done, so you should avoid modifying your filePro database while it may be opened in PHP.

## filepro\_fieldname

### Name

`filepro_fieldname` — gets the name of a field

### Description

```
string filepro_fieldname(int field_number);
```

Returns the name of the field corresponding to *field\_number*.

## filepro\_fieldtype

### Name

filepro\_fieldtype — gets the type of a field

### Description

```
string filepro_fieldtype(int field_number);
```

Returns the edit type of the field corresponding to *field\_number*.

## filepro\_fieldwidth

### Name

filepro\_fieldwidth — gets the width of a field

### Description

```
int filepro_fieldwidth(int field_number);
```

Returns the width of the field corresponding to *field\_number*.

## **filepro\_retrieve**

### **Name**

`filepro_retrieve` — retrieves data from a filePro database

### **Description**

```
string filepro_retrieve(int row_number, int field_number);
```

Returns the data from the specified location in the database.

## **filepro\_fieldcount**

### **Name**

`filepro_fieldcount` — find out how many fields are in a filePro database

### **Description**

```
int filepro_fieldcount(void);
```

Returns the number of fields (columns) in the opened filePro database.

See also `filepro`.

## **filepro\_rowcount**

### **Name**

`filepro_rowcount` — find out how many rows are in a filePro database

### **Description**

```
int filepro_rowcount(void);
```

Returns the number of rows in the opened filePro database.

See also `filepro`.

## **XVII. Filesystem functions**

# basename

## Name

basename — return filename component of path

## Description

```
string basename(string path);
```

Given a string containing a path to a file, this function will return the base name of the file.

On Windows, both slash (/) and backslash (\) are used as path separator character. In other environments, it is the forward slash (/).

### Example 1. **basename** example

```
$path = "/home/httpd/html/index.php3";  
$file = basename($path); // $file is set to "index.php3"
```

See also: `dirname`

# chgrp

## Name

chgrp — change file group

## Description

```
int chgrp(string filename, mixed group);
```

Attempts to change the group of the file *filename* to *group*. Only the superuser may change the group of a file arbitrarily; other users may change the group of a file to any group of which that user is a member.

Returns true on success; otherwise returns false.

On Windows, does nothing and returns true.

See also `chown` and `chmod`.

## chmod

### Name

`chmod` — change file mode

### Description

```
int chmod(string filename, int mode);
```

Attempts to change the mode of the file specified by *filename* to that given in *mode*.

Note that *mode* is not automatically assumed to be an octal value. To ensure the expected operation, you need to prefix *mode* with a zero (0):

```
chmod( "/somedir/somefile", 755 );    // decimal; probably incorrect  
chmod( "/somedir/somefile", 0755 );   // octal; correct value of mode
```

Returns true on success and false otherwise.

See also `chown` and `chgrp`.

# chown

## Name

chown — change file owner

## Description

```
int chown(string filename, mixed user);
```

Attempts to change the owner of the file filename to user user. Only the superuser may change the owner of a file.

Returns true on success; otherwise returns false.

**Note:** On Windows, does nothing and returns true.

See also `chown` and `chmod`.

# clearstatcache

## Name

clearstatcache — clear file stat cache

## Description

```
void clearstatcache(void);
```

Invoking the `stat` or `lstat` system call on most systems is quite expensive. Therefore, the result of the last call to any of the status functions (listed below) is stored for use on the next such call using the same

filename. If you wish to force a new status check, for instance if the file is being checked many times and may change or disappear, use this function to clear the results of the last call from memory.

This value is only cached for the lifetime of a single request.

Affected functions include `stat`, `lstat`, `file_exists`, `is_writeable`, `is_readable`, `is_executable`, `is_file`, `is_dir`, `is_link`, `filectime`, `fileatime`, `filemtime`, `fileinode`, `filegroup`, `fileowner`, `filesize`, `filetype`, and `fileperms`.

## copy

### Name

`copy` — copy file

### Description

```
int copy(string source, string dest);
```

Makes a copy of a file. Returns true if the copy succeeded, false otherwise.

#### Example 1. copy example

```
if (!copy($file, $file.'.bak')) {
    print("failed to copy $file...<br>\n");
}
```

See also: `rename`

# delete

## Name

`delete` — a dummy manual entry

## Description

```
void delete(string file);
```

This is a dummy manual entry to satisfy those people who are looking for `unlink` or `unset` in the wrong place.

See also: `unlink` to delete files, `unset` to delete variables.

# dirname

## Name

`dirname` — return directory name component of path

## Description

```
string dirname(string path);
```

Given a string containing a path to a file, this function will return the name of the directory.

On Windows, both slash (/) and backslash (\) are used as path separator character. In other environments, it is the forward slash (/).

### Example 1. `dirname` example

```
$path = "/etc/passwd";
```

```
$file = dirname($path); // $file is set to "/etc"
```

See also: `basename`

## diskfreespace

### Name

`diskfreespace` — return available space in directory

### Description

```
float diskfreespace(string directory);
```

Given a string containing a directory, this function will return the number of bytes available on the corresponding disk.

#### Example 1. `diskfreespace` example

```
$df = diskfreespace("/"); // $df contains the number of bytes available on "/"
```

## fclose

### Name

`fclose` — close an open file pointer

## Description

```
int fclose(int fp);
```

The file pointed to by *fp* is closed.

Returns true on success and false on failure.

The file pointer must be valid, and must point to a file successfully opened by *fopen* or *fsockopen*.

## **feof**

### Name

*feof* — test for end-of-file on a file pointer

## Description

```
int feof(int fp);
```

Returns true if the file pointer is at EOF or an error occurs; otherwise returns false.

The file pointer must be valid, and must point to a file successfully opened by *fopen*, *popen*, or *fsockopen*.

## **fgetc**

### Name

*fgetc* — get character from file pointer

## Description

```
string fgetc(int fp);
```

Returns a string containing a single character read from the file pointed to by `fp`. Returns FALSE on EOF (as does `feof`).

The file pointer must be valid, and must point to a file successfully opened by `fopen`, `popen`, or `fsockopen`.

See also `fread`, `fopen`, `popen`, `fsockopen`, and `fgets`.

## fgetcsv

### Name

`fgetcsv` — get line from file pointer and parse for CSV fields

## Description

```
array fgetcsv(int fp, int length, string [delimiter]);
```

Similar to `fgets()` except that `fgetcsv()` parses the line it reads for fields in CSV format and returns an array containing the fields read. The field delimiter is a comma, unless you specify another delimiter with the optional third parameter.

`fp` must be a valid file pointer to a file successfully opened by `fopen`, `popen`, or `fsockopen`

`length` must be greater than the longest line to be found in the CSV file (allowing for trailing line-end characters).

`fgetcsv()` returns false on error, including end of file.

NB A blank line in a CSV file will be returned as an array comprising just one single null field, and will not be treated as an error.

**Example 1. fgetcsv() example - Read and print entire contents of a CSV file**

```

$row=1;
$fp = fopen("test.csv","r");
while ($data = fgetcsv($fp,1000, ",")) {
    $num = count($data);
    print "<p> $num fields in line $row: <br>";
    $row++;
    for ( $c=0; $c<$num; $c++ ) print $data[$c] . "<br>";
}
fclose($fp);

```

## fgets

### Name

`fgets` — get line from file pointer

### Description

```
string fgets(int fp, int length);
```

Returns a string of up to `length - 1` bytes read from the file pointed to by `fp`. Reading ends when `length - 1` bytes have been read, on a newline (which is included in the return value), or on EOF (whichever comes first).

If an error occurs, returns false.

Common Pitfalls:

People used to the 'C' semantics of `fgets` should note the difference in how EOF is returned.

The file pointer must be valid, and must point to a file successfully opened by `fopen`, `popen`, or `fsockopen`.

A simple example follows:

**Example 1. Reading a file line by line**

```
$fd = fopen("/tmp/inputfile.txt", "r");
while (!feof($fd)) {
    $buffer = fgets($fd, 4096);
    echo $buffer;
}
fclose($fd);
```

See also `fread`, `fopen`, `popen`, `fgetc`, and `fsockopen`.

## fgetss

### Name

`fgetss` — get line from file pointer and strip HTML tags

### Description

```
string fgetss(int fp, int length, string [allowable_tags]);
```

Identical to `fgets`, except that `fgetss` attempts to strip any HTML and PHP tags from the text it reads.

You can use the optional third parameter to specify tags which should not be stripped.

**Note:** `allowable_tags` was added in PHP 3.0.13, PHP4B3.

See also `fgets`, `fopen`, `fsockopen`, `popen`, and `strip_tags`.

# file

## Name

`file` — read entire file into an array

## Description

```
array file(string filename, int [use_include_path]);
```

Identical to `readfile`, except that `file` returns the file in an array. Each element of the array corresponds to a line in the file, with the newline still attached.

You can use the optional second parameter and set it to "1", if you want to search for the file in the `include_path`, too.

See also `readfile`, `fopen`, and `popen`.

# file\_exists

## Name

`file_exists` — Check whether a file exists.

## Description

```
int file_exists(string filename);
```

Returns true if the file specified by *filename* exists; false otherwise.

The results of this function are cached. See `clearstatcache` for more details.

## fileatime

### Name

fileatime — get last access time of file

### Description

```
int fileatime(string filename);
```

Returns the time the file was last accessed, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## filectime

### Name

filectime — get inode change time of file

### Description

```
int filectime(string filename);
```

Returns the time the file was last changed, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## filegroup

### Name

filegroup — get file group

### Description

```
int filegroup(string filename);
```

Returns the group ID of the owner of the file, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## fileinode

### Name

fileinode — get file inode

### Description

```
int fileinode(string filename);
```

Returns the inode number of the file, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## filemtime

### Name

filemtime — get file modification time

### Description

```
int filemtime(string filename);
```

Returns the time the file was last modified, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## fileowner

### Name

fileowner — get file owner

### Description

```
int fileowner(string filename);
```

Returns the user ID of the owner of the file, or false in case of an

The results of this function are cached. See `clearstatcache` for more details. error.

## fileperms

### Name

fileperms — get file permissions

### Description

```
int fileperms(string filename);
```

Returns the permissions on the file, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## filesize

### Name

filesize — get file size

### Description

```
int filesize(string filename);
```

Returns the size of the file, or false in case of an error.

The results of this function are cached. See `clearstatcache` for more details.

## filetype

### Name

`filetype` — get file type

### Description

```
string filetype(string filename);
```

Returns the type of the file. Possible values are `fifo`, `char`, `dir`, `block`, `link`, `file`, and `unknown`.

Returns `false` if an error occurs.

The results of this function are cached. See `clearstatcache` for more details.

## flock

### Name

`flock` — portable advisory file locking

### Description

```
bool flock(int fp, int operation);
```

PHP supports a portable way of locking complete files in an advisory way (which means all accessing programs have to use the same way of locking or it will not work).

`flock` operates on `fp` which must be an open file pointer. `operation` is one of the following values:

- To acquire a shared lock (reader), set `operation` to 1.
- To acquire an exclusive lock (writer), set `operation` to 2.

- To release a lock (shared or exclusive), set *operation* to 3.
- If you don't want `flock` to block while locking, add 4 to *operation*.

`flock` allows you to perform a simple reader/writer model which can be used on virtually every platform (including most Unices and even Windows).

`flock` returns true on success and false on error (e.g. when a lock could not be acquired).

## fopen

### Name

`fopen` — open file or URL

### Description

```
int fopen(string filename, string mode, int [use_include_path]);
```

If *filename* begins with "http://" (not case sensitive), an HTTP 1.0 connection is opened to the specified server and a file pointer is returned to the beginning of the text of the response.

Does not handle HTTP redirects, so you must include trailing slashes on directories.

If *filename* begins with "ftp://" (not case sensitive), an ftp connection to the specified server is opened and a pointer to the requested file is returned. If the server does not support passive mode ftp, this will fail. You can open files for either reading and writing via ftp (but not both simultaneously).

If *filename* begins with anything else, the file will be opened from the filesystem, and a file pointer to the file opened is returned.

If the open fails, the function returns false.

*mode* may be any of the following:

- 'r' - Open for reading only; place the file pointer at the beginning of the file.
- 'r+' - Open for reading and writing; place the file pointer at the beginning of the file.
- 'w' - Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.

- 'w+' - Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
- 'a' - Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
- 'a+' - Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.

As well, *mode* may contain the letter 'b'. This is useful only on systems which differentiate between binary and text files (i.e., it's useless on Unix). If not needed, this will be ignored.

You can use the optional third parameter and set it to "1", if you want to search for the file in the `include_path`, too.

### Example 1. `fopen()` example

```
$fp = fopen("/home/rasmus/file.txt", "r");
$fp = fopen("http://www.php.net/", "r");
$fp = fopen("ftp://user:password@example.com/", "w");
```

If you are experiencing problems with reading and writing to files and you're using the server module version of PHP, remember to make sure that the files and directories you're using are accessible to the server process.

On the Windows platform, be careful to escape any backslashes used in the path to the file, or use forward slashes.

```
$fp = fopen("c:\\data\\info.txt", "r");
```

See also `fclose`, `fsockopen`, and `popen`.

## fpassthru

### Name

`fpassthru` — output all remaining data on a file pointer

## Description

```
int fpassthru(int fp);
```

Reads to EOF on the given file pointer and writes the results to standard output.

If an error occurs, `fpassthru` returns false.

The file pointer must be valid, and must point to a file successfully opened by `fopen`, `popen`, or `fsockopen`. The file is closed when `fpassthru` is done reading it (leaving *fp* useless).

If you just want to dump the contents of a file to stdout you may want to use the `readfile`, which saves you the `fopen` call.

See also `readfile`, `fopen`, `popen`, and `fsockopen`

## fputs

### Name

`fputs` — write to a file pointer

### Description

```
int fputs(int fp, string str, int [length]);
```

`fputs` is an alias to `fwrite`, and is identical in every way. Note that the *length* parameter is optional and if not specified the entire string will be written.

# fread

## Name

fread — Binary-safe file read

## Description

```
string fread(int fp, int length);
```

fread reads up to *length* bytes from the file pointer referenced by *fp*. Reading stops when *length* bytes have been read or EOF is reached, whichever comes first.

```
// get contents of a file into a string
$filename = "/usr/local/something.txt";
$fd = fopen( $filename, "r" );
$contents = fread( $fd, filesize( $filename ) );
fclose( $fd );
```

See also `fwrite`, `fopen`, `fsockopen`, `popen`, `fgets`, `fgetss`, `file`, and `fpassthru`.

# fseek

## Name

fseek — seek on a file pointer

## Description

```
int fseek(int fp, int offset);
```

Sets the file position indicator for the file referenced by `fp` to offset bytes into the file stream. Equivalent to calling (in C) `fseek( fp, offset, SEEK_SET )`.

Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

May not be used on file pointers returned by `fopen` if they use the "http://" or "ftp://" formats.

See also `ftell` and `rewind`.

## **ftell**

### **Name**

`ftell` — tell file pointer read/write position

### **Description**

```
int ftell(int fp);
```

Returns the position of the file pointer referenced by `fp`; i.e., its offset into the file stream.

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by `fopen` or `popen`.

See also `fopen`, `popen`, `fseek` and `rewind`.

## **fwrite**

### **Name**

`fwrite` — Binary-safe file write

## Description

```
int fwrite(int fp, string string, int [length]);
```

`fwrite` writes the contents of *string* to the file stream pointed to by *fp*. If the *length* argument is given, writing will stop after *length* bytes have been written or the end of *string* is reached, whichever comes first.

Note that if the *length* argument is given, then the `magic_quotes_runtime` configuration option will be ignored and no slashes will be stripped from *string*.

See also `fread`, `fopen`, `fsockopen`, `popen`, and `fputs`.

## set\_file\_buffer

### Name

`set_file_buffer` — Sets file buffering on the given file pointer

### Description

```
int fwrite(int fp, int buffer);
```

`set_file_buffer` sets the buffering for write operations on the given filepointer *fp* to *buffer* bytes. If *buffer* is 0 then write operations are unbuffered.

The function returns 0 on success, or EOF if the request cannot be honored.

Note that the default for any `fopen` with calling `set_file_buffer` is 8K.

See also `fopen`.

## is\_dir

### Name

`is_dir` — tells whether the filename is a directory

### Description

```
bool is_dir(string filename);
```

Returns true if the filename exists and is a directory.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_file` and `is_link`.

## is\_executable

### Name

`is_executable` — tells whether the filename is executable

### Description

```
bool is_executable(string filename);
```

Returns true if the filename exists and is executable.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_file` and `is_link`.

## is\_file

### Name

`is_file` — tells whether the filename is a regular file

### Description

```
bool is_file(string filename);
```

Returns true if the filename exists and is a regular file.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_dir` and `is_link`.

## is\_link

### Name

`is_link` — tells whether the filename is a symbolic link

### Description

```
bool is_link(string filename);
```

Returns true if the filename exists and is a symbolic link.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_dir` and `is_file`.

# is\_readable

## Name

`is_readable` — tells whether the filename is readable

## Description

```
bool is_readable(string filename);
```

Returns true if the filename exists and is readable.

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_writeable`.

# is\_writeable

## Name

`is_writeable` — tells whether the filename is writeable

## Description

```
bool is_writeable(string filename);
```

Returns true if the filename exists and is writeable. The filename argument may be a directory name allowing you to check if a directory is writeable.

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

The results of this function are cached. See `clearstatcache` for more details.

See also `is_readable`.

## link

### Name

`link` — Create a hard link

### Description

```
int link(string target, string link);
```

`Link` creates a hard link.

See also the `symlink` to create soft links, and `readlink` along with `linkinfo`.

## linkinfo

### Name

`linkinfo` — Get information about a link

### Description

```
int linkinfo(string path);
```

`Linkinfo` returns the `st_dev` field of the UNIX C `stat` structure returned by the `lstat` system call. This function is used to verify if a link (pointed to by `path`) really exists (using the same method as the `S_ISLNK` macro defined in `stat.h`). Returns 0 or `FALSE` in case of error.

See also `symlink`, `link`, and `readlink`.

## **mkdir**

### **Name**

`mkdir` — make directory

### **Description**

```
int mkdir(string pathname, int mode);
```

Attempts to create the directory specified by `pathname`.

Note that you probably want to specify the mode as an octal number, which means it should have a leading zero.

```
mkdir("/path/to/my/dir", 0700);
```

Returns true on success and false on failure.

See also `rmdir`.

## **pclose**

### **Name**

`pclose` — close process file pointer

## Description

```
int pclose(int fp);
```

Closes a file pointer to a pipe opened by `popen`.

The file pointer must be valid, and must have been returned by a successful call to `popen`.

Returns the termination status of the process that was run.

See also `popen`.

## popen

### Name

`popen` — open process file pointer

### Description

```
int popen(string command, string mode);
```

Opens a pipe to a process executed by forking the command given by `command`.

Returns a file pointer identical to that returned by `fopen`, except that it is unidirectional (may only be used for reading or writing) and must be closed with `pclose`. This pointer may be used with `fgets`, `fgetss`, and `fputs`.

If an error occurs, returns false.

```
$fp = popen( "/bin/ls", "r" );
```

See also `pclose`.

## readfile

### Name

`readfile` — output a file

### Description

```
int readfile(string filename, int [use_include_path]);
```

Reads a file and writes it to standard output.

Returns the number of bytes read from the file. If an error occurs, false is returned and unless the function was called as `@readfile`, an error message is printed.

If *filename* begins with "http://" (not case sensitive), an HTTP 1.0 connection is opened to the specified server and the text of the response is written to standard output.

Does not handle HTTP redirects, so you must include trailing slashes on directories.

If *filename* begins with "ftp://" (not case sensitive), an ftp connection to the specified server is opened and the requested file is written to standard output. If the server does not support passive mode ftp, this will fail.

If *filename* begins with neither of these strings, the file will be opened from the filesystem and its contents written to standard output.

You can use the optional second parameter and set it to "1", if you want to search for the file in the `include_path`, too.

See also `fpasssthru`, `file`, `fopen`, `include`, `require`, and `virtual`.

## readlink

### Name

`readlink` — Return the target of a symbolic link

## Description

```
string readlink(string path);
```

Readlink does the same as the readlink C function and returns the contents of the symbolic link path or 0 in case of error.

See also symlink, readlink and linkinfo.

## rename

### Name

rename — rename a file

### Description

```
int rename(string oldname, string newname);
```

Attempts to rename *oldname* to *newname*.

Returns true on success and false on failure.

## rewind

### Name

rewind — rewind the position of a file pointer

## Description

```
int rewind(int fp);
```

Sets the file position indicator for *fp* to the beginning of the file stream.

If an error occurs, returns 0.

The file pointer must be valid, and must point to a file successfully opened by *fopen*.

See also *fseek* and *ftell*.

## rmdir

### Name

*rmdir* — remove directory

### Description

```
int rmdir(string dirname);
```

Attempts to remove the directory named by *pathname*. The directory must be empty, and the relevant permissions must permit this.

If an error occurs, returns 0.

See also *mkdir*.

## stat

### Name

*stat* — give information about a file

## Description

```
array stat(string filename);
```

Gathers the statistics of the file named by filename.

Returns an array with the statistics of the file with the following elements:

1. device
2. inode
3. inode protection mode
4. number of links
5. user id of owner
6. group id owner
7. device type if inode device \*
8. size in bytes
9. time of last access
10. time of last modification
11. time of last change
12. blocksize for filesystem I/O \*
13. number of blocks allocated

\* - only valid on systems supporting the st\_blksize type—other systems (i.e. Windows) return -1

The results of this function are cached. See `clearstatcache` for more details.

## Istat

### Name

`lstat` — give information about a file or symbolic link

## Description

```
array lstat(string filename);
```

Gathers the statistics of the file or symbolic link named by *filename*. This function is identical to the `stat` function except that if the *filename* parameter is a symbolic link, the status of the symbolic link is returned, not the status of the file pointed to by the symbolic link.

Returns an array with the statistics of the file with the following elements:

1. device
  2. inode
  3. number of links
  4. user id of owner
  5. group id owner
  6. device type if inode device \*
  7. size in bytes
  8. time of last access
  9. time of last modification
  10. time of last change
  11. blocksize for filesystem I/O \*
  12. number of blocks allocated
- \* - only valid on systems supporting the `st_blksize` type—other systems (i.e. Windows) return -1

The results of this function are cached. See `clearstatcache` for more details.

## symlink

### Name

`symlink` — Create a symbolic link

## Description

```
int symlink(string target, string link);
```

`symlink` creates a symbolic link from the existing *target* with the specified name *link*.

See also `link` to create hard links, and `readlink` along with `linkinfo`.

## tempnam

### Name

`tempnam` — create unique file name

### Description

```
string tempnam(string dir, string prefix);
```

Creates a unique temporary filename in the specified directory. If the directory does not exist, `tempnam` may generate a filename in the system's temporary directory.

The behaviour of the `tempnam` function is system dependent. On Windows the `TMP` environment variable will override the *dir* parameter, on Linux the `TMPDIR` environment variable has precedence, while SVR4 will always use your *dir* parameter if the directory it points to exists. Consult your system documentation on the `tempnam(3)` function if in doubt.

Returns the new temporary filename, or the null string on failure.

#### Example 1. `tempnam()` example

```
$tmpfname = tempnam( "/tmp", "FOO" );
```

## touch

### Name

`touch` — set modification time of file

### Description

```
int touch(string filename, int time);
```

Attempts to set the modification time of the file named by `filename` to the value given by `time`. If the option `time` is not given, uses the present time.

If the file does not exist, it is created.

Returns true on success and false otherwise.

## umask

### Name

`umask` — changes the current umask

### Description

```
int umask(int mask);
```

`Umask` sets PHP's umask to `mask & 0777` and returns the old umask. When PHP is being used as a server module, the umask is restored when each request is finished.

`Umask` without arguments simply returns the current umask.

# unlink

## Name

unlink — Delete a file

## Description

```
int unlink(string filename);
```

Deletes *filename*. Similar to the Unix C unlink() function.

Returns 0 or FALSE on an error.

See also `rmdir` for removing directories.

## **XVIII. FTP functions**

FTP stands for File Transfer Protocol.

The following constants are defined when using the FTP module: `FTP_ASCII`, and `FTP_BINARY`.

## ftp\_connect

### Name

`ftp_connect` — Opens up an FTP connection

### Description

```
int ftp_connect(string host, int [port] );
```

Returns a FTP stream on success, false on error.

`ftp_connect` opens up a FTP connection to the specified *host*. The *port* parameter specifies an alternate port to connect to. If it is omitted or zero, then the default FTP port, 21, will be used.

## ftp\_login

### Name

`ftp_login` — Logs in an FTP connection

### Description

```
int ftp_login(int ftp_stream, string username, string password);
```

Returns true on success, false on error.

Logs in the given FTP stream.

## ftp\_pwd

### Name

`ftp_pwd` — Returns the current directory name

### Description

```
int ftp_pwd(int ftp_stream);
```

Returns the current directory, or false on error.

## ftp\_cdup

### Name

`ftp_cdup` — Changes to the parent directory

### Description

```
int ftp_cdup(int ftp_stream);
```

Returns true on success, false on error.

Changes to the parent directory.

## ftp\_chdir

### Name

`ftp_chdir` — Changes directories on a FTP server

### Description

```
int ftp_chdir(int ftp_stream, string directory);
```

Returns true on success, false on error.

Changes to the specified *directory*.

## ftp\_mkdir

### Name

`ftp_mkdir` — Creates a directory

### Description

```
string ftp_mkdir(int ftp_stream, string directory);
```

Returns the newly created directory name on success, false on error.

Creates the specified *directory*.

## ftp\_rmdir

### Name

`ftp_rmdir` — Removes a directory

### Description

```
int ftp_rmdir(int ftp_stream, string directory);
```

Returns true on success, false on error.

Removes the specified *directory*.

## ftp\_nlist

### Name

`ftp_nlist` — Returns a list of files in the given directory.

### Description

```
int ftp_nlist(int ftp_stream, string directory);
```

Returns an array of filenames on success, false on error.

## ftp\_rawlist

### Name

`ftp_rawlist` — Returns a detailed list of files in the given directory.

### Description

```
int ftp_rawlist(int ftp_stream, string directory);
```

`ftp_rawlist` executes the FTP LIST command, and returns the result as an array. Each array element corresponds to one line of text. The output is not parsed in any way. The system type identifier returned by `ftp_systype` can be used to determine how the results should be interpreted.

## ftp\_systype

### Name

`ftp_systype` — Returns the system type identifier of the remote FTP server.

### Description

```
int ftp_systype(int ftp_stream);
```

Returns the remote system type, or false on error.

## ftp\_pasv

### Name

`ftp_pasv` — Turns passive mode on or off.

### Description

```
int ftp_pasv(int ftp_stream, int pasv);
```

Returns true on success, false on error.

`ftp_pasv` turns on passive mode if the *pasv* parameter is true (it turns off passive mode if *pasv* is false.) In passive mode, data connections are initiated by the client, rather than by the server.

## ftp\_get

### Name

`ftp_get` — Downloads a file from the FTP server.

### Description

```
int ftp_get(int ftp_stream, string local_file, string remote_file, int mode);
```

Returns true on success, false on error.

`ftp_get` retrieves *remote\_file* from the FTP server, and saves it to *local\_file* locally. The transfer *mode* specified must be either `FTP_ASCII` or `FTP_BINARY`.

## ftp\_fget

### Name

`ftp_fget` — Downloads a file from the FTP server and saves to an open file.

### Description

```
int ftp_fget(int ftp_stream, int fp, string remote_file, int mode);
```

Returns true on success, false on error.

`ftp_fget` retrieves *remote\_file* from the FTP server, and writes it to the given file pointer, *fp*. The transfer *mode* specified must be either `FTP_ASCII` or `FTP_BINARY`.

## ftp\_put

### Name

`ftp_put` — Uploads a file to the FTP server.

### Description

```
int ftp_put(int ftp_stream, string remote_file, string local_file, int mode);
```

Returns true on success, false on error.

`ftp_put` stores *local\_file* on the FTP server, as *remote\_file*. The transfer *mode* specified must be either `FTP_ASCII` or `FTP_BINARY`.

## ftp\_fput

### Name

`ftp_fput` — Uploads from an open file to the FTP server.

### Description

```
int ftp_fput(int ftp_stream, string remote_file, int fp, int mode);
```

Returns true on success, false on error.

`ftp_fput` uploads the data from the file pointer *fp* until end of file. The results are stored in *remote\_file* on the FTP server. The transfer *mode* specified must be either `FTP_ASCII` or `FTP_BINARY`.

## ftp\_size

### Name

`ftp_size` — Returns the size of the given file.

### Description

```
int ftp_size(int ftp_stream, string remote_file);
```

Returns the file size on success, or -1 on error.

`ftp_size` returns the size of a file. If an error occurs, or if the file does not exist, -1 is returned. Not all servers support this feature.

## ftp\_mdtm

### Name

`ftp_mdtm` — Returns the last modified time of the given file.

### Description

```
int ftp_mdtm(int ftp_stream, string remote_file);
```

Returns a UNIX timestamp on success, or -1 on error.

`ftp_mdtm` checks the last-modified time for a file, and returns it as a UNIX timestamp. If an error occurs, or the file does not exist, -1 is returned. Note that not all servers support this feature.

## ftp\_rename

### Name

`ftp_rename` — Renames a file on the ftp server.

### Description

```
int ftp_rename(int ftp_stream, string from, string to);
```

Returns true on success, false on error.

`ftp_rename` renames the file specified by *from* to the new name *to*

## ftp\_delete

### Name

`ftp_delete` — Deletes a file on the ftp server.

### Description

```
int ftp_delete(int ftp_stream, string path);
```

Returns true on success, false on error.

`ftp_delete` deletes the file specified by *path* from the FTP server.

## ftp\_quit

### Name

`ftp_quit` — Closes an FTP connection

### Description

```
int ftp_quit(int ftp_stream);
```

`ftp_quit` closes *ftp\_stream*.

## **XIX. HTTP functions**

These functions let you manipulate the output sent back to the remote browser right down to the HTTP protocol level.

# header

## Name

header — Send a raw HTTP header

## Description

```
int header(string string);
```

The `Header` function is used at the top of an HTML file to send raw HTTP header strings. See the HTTP 1.1 Specification (<http://www.w3.org/Protocols/rfc2068/rfc2068>) for more information on raw http headers. *Note:* Remember that the `Header` function must be called before any actual output is sent either by normal HTML tags or from PHP. It is a very common error to read code with `include` or with `auto_prepend` and have spaces or empty lines in this code that force output before `header` is called.

```
header("Location: http://www.php.net"); /* Redi-  
rect browser to PHP web site */  
exit; /* Make sure that code below does not get executed when we redi-  
rect. */
```

PHP scripts often generate dynamic HTML that must not be cached by the client browser or any proxy caches between the server and the client browser. Many proxies and clients can be forced to disable caching with

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT"); // Date in the past  
header("Last-  
Modified: " . gmdate("D, d M Y H:i:s") . " GMT"); // always modified  
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1  
header("Pragma: no-cache"); // HTTP/1.0
```

# setcookie

## Name

setcookie — Send a cookie

## Description

```
int setcookie(string name, string value, int expire, string path, string
domain, int secure);
```

`setcookie` defines a cookie to be sent along with the rest of the header information. Cookies must be sent *before* any other headers are sent (this is a restriction of cookies, not PHP). This requires you to place calls to this function before any `<html>` or `<head>` tags.

All the arguments except the *name* argument are optional. If only the name argument is present, the cookie by that name will be deleted from the remote client. You may also replace any argument with an empty string (`""`) in order to skip that argument. The *expire* and *secure* arguments are integers and cannot be skipped with an empty string. Use a zero (`0`) instead. The *expire* argument is a regular Unix time integer as returned by the `time` or `mktime` functions. The *secure* indicates that the cookie should only be transmitted over a secure HTTPS connection.

Common Pitfalls:

Cookies will not become visible until the next loading of a page that the cookie should be visible for.

Multiple calls to `setcookie` in the same script will be performed in the reverse order. If you are trying to delete one cookie before inserting another you should put the insert before the delete.

Some examples follow:

### Example 1. `setcookie` examples

```
setcookie("TestCookie","Test Value");
setcookie("TestCookie",$value,time()+3600); /* expire in 1 hour */
setcookie("TestCookie",$value,time()+3600,"/~rasmus/",".utoronto.ca",1);
```

Note that the value portion of the cookie will automatically be urlencoded when you send the cookie, and when it is received, it is automatically decoded and assigned to a variable by the same name as the cookie name. To see the contents of our test cookie in a script, simply use one of the following examples:

```
echo $TestCookie;  
echo $HTTP_COOKIE_VARS[ "TestCookie" ];
```

For more information on cookies, see Netscape's cookie specification at [http://www.netscape.com/newsref/std/cookie\\_spec.html](http://www.netscape.com/newsref/std/cookie_spec.html).

Microsoft Internet Explorer 4 with Service Pack 1 applied does not correctly deal with cookies that have their path parameter set.

Netscape Communicator 4.05 and Microsoft Internet Explorer 3.x appear to handle cookies incorrectly when the path and time are not set.

# XX. Hyperwave functions

## Introduction

Hyperwave has been developed at IICM (<http://www.iicm.edu>) in Graz. It started with the name Hyper-G and changed to Hyperwave when it was commercialised (If I remember properly it was in 1996).

Hyperwave is not free software. The current version, 4.1, is available at [www.hyperwave.com](http://www.hyperwave.com) (<http://www.hyperwave.com/>). A time limited version can be ordered for free (30 days).

Hyperwave is an information system similar to a database (HIS, Hyperwave Information Server). Its focus is the storage and management of documents. A document can be any possible piece of data that may as well be stored in file. Each document is accompanied by its object record. The object record contains meta data for the document. The meta data is a list of attributes which can be extended by the user. Certain attributes are always set by the Hyperwave server, other may be modified by the user. An attribute is a name/value pair of the form name=value. The complete object record contains as many of those pairs as the user likes. The name of an attribute does not have to be unique, e.g. a title may appear several times within an object record. This makes sense if you want to specify a title in several languages. In such a case there is a convention, that each title value is preceded by the two letter language abbreviation followed by a colon, e.g. 'en:Title in English' or 'ge:Titel in deutsch'. Other attributes like a description or keywords are potential candidates. You may also replace the language abbreviation by any other string as long as it separated by colon from the rest of the attribute value.

Each object record has native a string representation with each name/value pair separated by a newline. The Hyperwave extension also knows a second representation which is an associated array with the attribute name being the key. Multilingual attribute values itself form another associated array with the key being the language abbreviation. Actually any multiple attribute forms an associated array with the string left to the colon in the attribute value being the key. (This is not fully implemented. Only the attributes Title, Description and Keyword are treated properly yet.)

Besides the documents, all hyper links contained in a document are stored as object records as well. Hyper links which are in a document will be removed from it and stored as individual objects, when the document is inserted into the database. The object record of the link contains information about where it starts and where it ends. In order to gain the original document you will have to retrieve the plain document without the links and the list of links and reinsert them (The functions `hw_pipedocument` and `hw_gettext` do this for you. The advantage of separating links from the document is obvious. Once a document to which a link is pointing to changes its name, the link can easily be modified accordingly. The document containing the link is not affected at all. You may even add a link to a document without modifying the document itself.

Saying that `hw_pipedocument` and `hw_gettext` do the link insertion automatically is not as simple as it sounds. Inserting links implies a certain hierarchy of the documents. On a web server this is given by

the file system, but Hyperwave has its own hierarchy and names do not reflect the position of an object in that hierarchy. Therefore creation of links first of all requires a mapping from the Hyperwave hierarchy and namespace into a web hierarchy respective web namespace. The fundamental difference between Hyperwave and the web is the clear distinction between names and hierarchy in Hyperwave. The name does not contain any information about the objects position in the hierarchy. In the web the name also contains the information on where the object is located in the hierarchy. This leads to two possible ways of mapping. Either the Hyperwave hierarchy and name of the Hyperwave object is reflected in the URL or the name only. To make things simple the second approach is used. Hyperwave object with name 'my\_object' is mapped to 'http://host/my\_object' disregarding where it resides in the Hyperwave hierarchy. An object with name 'parent/my\_object' could be the child of 'my\_object' in the Hyperwave hierarchy, though in a web namespace it appears to be just the opposite and the user might get confused. This can only be prevented by selecting reasonable object names.

Having made this decision a second problem arises. How do you involve PHP? The URL `http://host/my_object` will not call any PHP script unless you tell your web server to rewrite it to e.g. `'http://host/php3_script/my_object'` and the script `'php3_script'` evaluates the `$PATH_INFO` variable and retrieves the object with name 'my\_object' from the Hyperwave server. There is just one little drawback which can be fixed easily. Rewriting any URL would not allow any access to other document on the web server. A PHP script for searching in the Hyperwave server would be impossible. Therefore you will need at least a second rewriting rule to exclude certain URLs like all e.g. starting with `http://host/Hyperwave`. This is basically sharing of a namespace by the web and Hyperwave server.

Based on the above mechanism links are inserted into documents.

It gets more complicated if PHP is not run as a server module or CGI script but as a standalone application e.g. to dump the content of the Hyperwave server on a CD-ROM. In such a case it makes sense to retain the Hyperwave hierarchy and map it onto the file system. This conflicts with the object names if they reflect its own hierarchy (e.g. by choosing names including '/'). Therefore '/' has to be replaced by another character, e.g. '\_'. to be continued.

The network protocol to communicate with the Hyperwave server is called HG-CSP (<http://www.hyperwave.de/7.17-hg-prot>) (Hyper-G Client/Server Protocol). It is based on messages to initiate certain actions, e.g. get object record. In early versions of the Hyperwave Server two native clients (Harmony, Amadeus) were provided for communication with the server. Those two disappeared when Hyperwave was commercialised. As a replacement a so called wavemaster was provided. The wavemaster is like a protocol converter from HTTP to HG-CSP. The idea is to do all the administration of the database and visualisation of documents by a web interface. The wavemaster implements a set of placeholders for certain actions to customise the interface. This set of placeholders is called the PLACE Language. PLACE lacks a lot of features of a real programming language and any extension to it only enlarges the list of placeholders. This has led to the use of JavaScript which IMO does not make life easier.

Adding Hyperwave support to PHP should fill in the gap of a missing programming language for

interface customisation. It implements all the messages as defined by the HG-CSP but also provides more powerful commands to e.g. retrieve complete documents.

Hyperwave has its own terminology to name certain pieces of information. This has widely been taken over and extended. Almost all functions operate on one of the following data types.

- **object ID:** An unique integer value for each object in the Hyperwave server. It is also one of the attributes of the object record (ObjectID). Object ids are often used as an input parameter to specify an object.
- **object record:** A string with attribute-value pairs of the form attribute=value. The pairs are separated by a carriage return from each other. An object record can easily be converted into an object array with `hw_object2array`. Several functions return object records. The names of those functions end with `obj`.
- **object array:** An associated array with all attributes of an object. The key is the attribute name. If an attribute occurs more than once in an object record it will result in another indexed or associated array. Attributes which are language depended (like the title, keyword, description) will form an associated array with the key set to the language abbreviation. All other multiple attributes will form an indexed array. PHP functions never return object arrays.
- **hw\_document:** This is a complete new data type which holds the actual document, e.g. HTML, PDF etc. It is somewhat optimised for HTML documents but may be used for any format.

Several functions which return an array of object records do also return an associated array with statistical information about them. The array is the last element of the object record array. The statistical array contains the following entries:

#### Hidden

Number of object records with attribute PresentationHints set to Hidden.

#### CollectionHead

Number of object records with attribute PresentationHints set to CollectionHead.

#### FullCollectionHead

Number of object records with attribute PresentationHints set to FullCollectionHead.

#### CollectionHeadNr

Index in array of object records with attribute PresentationHints set to CollectionHead.

FullCollectionHeadNr

Index in array of object records with attribute PresentationHints set to FullCollectionHead.

Total

Total: Number of object records.

## Integration with Apache

The Hyperwave extension is best used when PHP is compiled as an Apache module. In such a case the underlying Hyperwave server can be hidden from users almost completely if Apache uses its rewriting engine. The following instructions will explain this.

Since PHP with Hyperwave support built into Apache is intended to replace the native Hyperwave solution based on Wavemaster I will assume that the Apache server will only serve as a Hyperwave web interface. This is not necessary but it simplifies the configuration. The concept is quite simple. First of all you need a PHP script which evaluates the PATH\_INFO variable and treats its value as the name of a Hyperwave object. Let's call this script 'Hyperwave'. The URL `http://your.hostname/Hyperwave/name_of_object` would then return the Hyperwave object with the name 'name\_of\_object'. Depending on the type of the object the script has to react accordingly. If it is a collection, it will probably return a list of children. If it is a document it will return the mime type and the content. A slight improvement can be achieved if the Apache rewriting engine is used. From the users point of view it would be more straight forward if the URL `http://your.hostname/name_of_object` would return the object. The rewriting rule is quite easy:

```
RewriteRule ^/(.*) /usr/local/apache/htdocs/HyperWave/$1 [L]
```

Now every URL relates to an object in the Hyperwave server. This causes a simple to solve problem. There is no way to execute a different script, e.g. for searching, than the 'Hyperwave' script. This can be fixed with another rewriting rule like the following:

```
RewriteRule ^/hw/(.*) /usr/local/apache/htdocs/hw/$1 [L]
```

This will reserve the directory `/usr/local/apache/htdocs/hw` for additional scripts and other files. Just make sure this rule is evaluated before the one above. There is just a little drawback: all Hyperwave objects whose name starts with 'hw/' will be shadowed. So, make sure you don't use such names. If you need more directories, e.g. for images just add more rules or place them all in one directory. Finally, don't forget to turn on the rewriting engine with

RewriteEngine on

My experiences have shown that you will need the following scripts:

- to return the object itself
- to allow searching
- to identify yourself
- to set your profile
- one for each additional function like to show the object attributes, to show information about users, to show the status of the server, etc.

## Todo

There are still some things todo:

- The `hw_InsertDocument` has to be split into `hw_InsertObject` and `hw_PutDocument`.
- The names of several functions are not fixed, yet.
- Most functions require the current connection as its first parameter. This leads to a lot of typing, which is quite often not necessary if there is just one open connection. A default connection will improve this.
- Conversion from object record into object array needs to handle any multiple attribute.

## hw\_Array2Objrec

### Name

hw\_Array2Objrec — convert attributes from object array to object record

### Description

```
strin hw_array2objrec (array object_array);
```

Converts an *object\_array* into an object record. Multiple attributes like 'Title' in different languages are treated properly.

See also hw\_objrec2array.

## hw\_Children

### Name

hw\_Children — object ids of children

### Description

```
array hw_children (int connection, int objectID);
```

Returns an array of object ids. Each id belongs to a child of the collection with ID *objectID*. The array contains all children both documents and collections.

## hw\_ChildrenObj

### Name

hw\_ChildrenObj — object records of children

### Description

```
array hw_childrenobj(int connection, int objectID);
```

Returns an array of object records. Each object record belongs to a child of the collection with ID *objectID*. The array contains all children both documents and collections.

## hw\_Close

### Name

hw\_Close — closes the Hyperwave connection

### Description

```
int hw_close(int connection);
```

Returns false if connection is not a valid connection index, otherwise true. Closes down the connection to a Hyperwave server with the given connection index.

## hw\_Connect

### Name

hw\_Connect — opens a connection

### Description

```
int hw_connect(string host, int port, string username, string password);
```

Opens a connection to a Hyperwave server and returns a connection index on success, or false if the connection could not be made. Each of the arguments should be a quoted string, except for the port number. The *username* and *password* arguments are optional and can be left out. In such a case no identification with the server will be done. It is similar to identify as user anonymous. This function returns a connection index that is needed by other Hyperwave functions. You can have multiple connections open at once. Keep in mind, that the password is not encrypted.

See also hw\_pConnect.

## hw\_Cp

### Name

hw\_Cp — copies objects

### Description

```
int hw_cp(int connection, array object_id_array, int destination id);
```

Copies the objects with object ids as specified in the second parameter to the collection with the id *destination id*.

The value return is the number of copied objects.

See also `hw_mv`.

## hw\_Deleteobject

### Name

`hw_Deleteobject` — deletes object

### Description

```
int hw_deleteobject (int connection, int object_to_delete);
```

Deletes the object with the given object id in the second parameter. It will delete all instances of the object.

Returns TRUE if no error occurs otherwise FALSE.

See also `hw_mv`.

## hw\_DocByAnchor

### Name

`hw_DocByAnchor` — object id object belonging to anchor

### Description

```
int hw_docbyanchor (int connection, int anchorID);
```

Returns an th object id of the document to which *anchorID* belongs.

## hw\_DocByAnchorObj

### Name

`hw_DocByAnchorObj` — object record object belonging to anchor

### Description

```
string hw_docbyanchorobj(int connection, int anchorID);
```

Returns an th object record of the document to which *anchorID* belongs.

## hw\_DocumentAttributes

### Name

`hw_DocumentAttributes` — object record of hw\_document

### Description

```
string hw_documentattributes(int hw_document);
```

Returns the object record of the document.

See also `hw_DocumentBodyTag`, `hw_DocumentSize`.

## hw\_DocumentBodyTag

### Name

hw\_DocumentBodyTag — body tag of hw\_document

### Description

```
string hw_documentbodytag (int hw_document);
```

Returns the BODY tag of the document. If the document is an HTML document the BODY tag should be printed before the document.

See also hw\_DocumentAttributes, hw\_DocumentSize.

## hw\_DocumentContent

### Name

hw\_DocumentContent — returns content of hw\_document

### Description

```
string hw_documentcontent (int hw_document);
```

Returns the content of the document. If the document is an HTML document the content is everything after the BODY tag. Information from the HEAD and BODY tag is in the stored in the object record.

See also hw\_DocumentAttributes, hw\_DocumentSize, hw\_DocumentSetContent.

## hw\_DocumentSetContent

### Name

`hw_DocumentSetContent` — sets/replaces content of `hw_document`

### Description

```
string hw_documentsetcontent (int hw_document, string content);
```

Sets or replaces the content of the document. If the document is an HTML document the content is everything after the BODY tag. Information from the HEAD and BODY tag is stored in the object record. If you provide this information in the content of the document too, the Hyperwave server will change the object record accordingly when the document is inserted. Probably not a very good idea. If this function fails the document will retain its old content.

See also `hw_DocumentAttributes`, `hw_DocumentSize`, `hw_DocumentContent`.

## hw\_DocumentSize

### Name

`hw_DocumentSize` — size of `hw_document`

### Description

```
int hw_documentsize (int hw_document);
```

Returns the size in bytes of the document.

See also `hw_DocumentBodyTag`, `hw_DocumentAttributes`.

## hw\_ErrorMsg

### Name

hw\_ErrorMsg — returns error message

### Description

```
string hw_errormsg(int connection);
```

Returns a string containing the last error message or 'No Error'. If false is returned, this function failed. The message relates to the last command.

## hw\_EditText

### Name

hw\_EditText — retrieve text document

### Description

```
int hw_edittest(int connection, int hw_document);
```

Uploads the text document to the server. The object record of the document may not be modified while the document is edited. This function will only works for pure text documents. It will not open a special data connection and therefore blocks the control connection during the transfer.

See also hw\_PipeDocument, hw\_FreeDocument, hw\_DocumentBodyTag, hw\_DocumentSize, hw\_OutputDocument, hw\_GetText.

## hw\_Error

### Name

hw\_Error — error number

### Description

```
int hw_error(int connection);
```

Returns the last error number. If the return value is 0 no error has occurred. The error relates to the last command.

## hw\_Free\_Document

### Name

hw\_Free\_Document — frees hw\_document

### Description

```
int hw_free_document(int hw_document);
```

Frees the memory occupied by the Hyperwave document.

## hw\_GetParents

### Name

`hw_GetParents` — object ids of parents

### Description

```
array hw_getparentsobj(int connection, int objectID);
```

Returns an indexed array of object ids. Each object id belongs to a parent of the object with ID *objectID*.

## hw\_GetParentsObj

### Name

`hw_GetParentsObj` — object records of parents

### Description

```
array hw_getparentsobj(int connection, int objectID);
```

Returns an indexed array of object records plus an associated array with statistical information about the object records. The associated array is the last entry of the returned array. Each object record belongs to a parent of the object with ID *objectID*.

## hw\_GetChildColl

### Name

hw\_GetChildColl — object ids of child collections

### Description

```
array hw_getchildcoll(int connection, int objectID);
```

Returns an array of object ids. Each object ID belongs to a child collection of the collection with ID *objectID*. The function will not return child documents.

See also hw\_GetChildren, hw\_GetChildDocColl.

## hw\_GetChildCollObj

### Name

hw\_GetChildCollObj — object records of child collections

### Description

```
array hw_getchildcollobj(int connection, int objectID);
```

Returns an array of object records. Each object records belongs to a child collection of the collection with ID *objectID*. The function will not return child documents.

See also hw\_ChildrenObj, hw\_GetChildDocCollObj.

## hw\_GetRemote

### Name

hw\_GetRemote — Gets a remote document

### Description

```
int hw_getremote(int connection, int objectID);
```

Returns a remote document. Remote documents in Hyperwave notation are documents retrieved from an external source. Common remote documents are for example external web pages or queries in a database. In order to be able to access external sources through remote documents Hyperwave introduces the HGI (Hyperwave Gateway Interface) which is similar to the CGI. Currently, only ftp, http-servers and some databases can be accessed by the HGI. Calling `hw_GetRemote` returns the document from the external source. If you want to use this function you should be very familiar with HGIs. You should also consider to use PHP instead of Hyperwave to access external sources. Adding database support by a Hyperwave gateway should be more difficult than doing it in PHP.

See also `hw_GetRemoteChildren`.

## hw\_GetRemoteChildren

### Name

hw\_GetRemoteChildren — Gets children of remote document

### Description

```
int hw_getremotechildren(int connection, string object record);
```

Returns the children of a remote document. Children of a remote document are remote documents itself. This makes sense if a database query has to be narrowed and is explained in Hyperwave Programmers'

Guide. If the number of children is 1 the function will return the document itself formatted by the Hyperwave Gateway Interface (HGI). If the number of children is greater than 1 it will return an array of object record with each maybe the input value for another call to `hw_GetRemoteChildren`. Those object records are virtual and do not exist in the Hyperwave server, therefore they do not have a valid object ID. How exactly such an object record looks like is up to the HGI. If you want to use this function you should be very familiar with HGIs. You should also consider to use PHP instead of Hyperwave to access external sources. Adding database support by a Hyperwave gateway should be more difficult than doing it in PHP.

See also `hw_GetRemote`.

## hw\_GetSrcByDestObj

### Name

`hw_GetSrcByDestObj` — Returns anchors pointing at object

### Description

```
array hw_getsrcbydestobj (int connection, int objectID);
```

Returns the object records of all anchors pointing to the object with ID *objectID*. The object can either be a document or an anchor of type destination.

See also `hw_GetAnchors`.

## hw\_GetObject

### Name

`hw_GetObject` — object record

## Description

```
array hw_getobject (int connection, [int|array] objectID, string query);
```

Returns the object record for the object with ID *objectID* if the second parameter is an integer. If the second parameter is an array of integer the function will return an array of object records. In such a case the last parameter is also evaluated which is a query string.

The query string has the following syntax:

```
<expr> ::= "(" <expr> ")" |
"!" <expr> /* NOT */
<expr> "||" <expr> /* OR */
<expr> "&&" <expr> /* AND */
<attribute> <operator> <value>
<attribute> ::= /* any attribute name (Title, Author, DocumentType ...) */
<operator> ::= "=" /* equal */
"<" /* less than (string compare) */
">" /* greater than (string compare) */
"~" /* regular expression matching */
```

The query allows to further select certain objects from the list of given objects. Unlike the other query functions, this query may use not indexed attributes. How many object records are returned depends on the query and if access to the object is allowed.

See also `hw_GetAndLock`, `hw_GetObjectByQuery`.

## hw\_GetAndLock

### Name

`hw_GetAndLock` — return object record and lock object

## Description

```
string hw_getandlock(int connection, int objectID);
```

Returns the object record for the object with ID *objectID*. It will also lock the object, so other users cannot access it until it is unlocked.

See also `hw_Unlock`, `hw_GetObject`.

## hw\_GetText

### Name

`hw_GetText` — retrieve text document

## Description

```
int hw_gettext(int connection, int objectID, mixed [rootID/prefix] );
```

Returns the document with object ID *objectID*. If the document has anchors which can be inserted, they will be inserted already. The optional parameter *rootID/prefix* can be a string or an integer. If it is an integer it determines how links are inserted into the document. The default is 0 and will result in links that are constructed from the name of the link's destination object. This is useful for web applications. If a link points to an object with name 'internet\_movie' the HTML link will be <A HREF="/internet\_movie">. The actual location of the source and destination object in the document hierarchy is disregarded. You will have to set up your web browser, to rewrite that URL to for example '/my\_script.php3/internet\_movie'. 'my\_script.php3' will have to evaluate \$PATH\_INFO and retrieve the document. All links will have the prefix '/my\_script.php3/'. If you do not want this you can set the optional parameter *rootID/prefix* to any prefix which is used instead. In this case it has to be a string.

If *rootID/prefix* is an integer and unequal to 0 the link is constructed from all the names starting at the object with the id *rootID/prefix* separated by a slash relative to the current object. If for example the above document 'internet\_movie' is located at 'a-b-c-internet\_movie' with '-' being the separator between hierarchy levels on the Hyperwave server and the source document is located at 'a-b-d-source' the resulting HTML link would be: <A HREF="../c/internet\_movie">. This is useful if

you want to download the whole server content onto disk and map the document hierarchy onto the file system.

This function will only work for pure text documents. It will not open a special data connection and therefore blocks the control connection during the transfer.

See also `hw_PipeDocument`, `hw_FreeDocument`, `hw_DocumentBodyTag`, `hw_DocumentSize`, `hw_OutputDocument`.

## hw\_GetObjectByQuery

### Name

`hw_GetObjectByQuery` — search object

### Description

```
array hw_getobjectbyquery (int connection, string query, int max_hits);
```

Searches for objects on the whole server and returns an array of object ids. The maximum number of matches is limited to *max\_hits*. If *max\_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also `hw_GetObjectByQueryObj`.

## hw\_GetObjectByQueryObj

### Name

`hw_GetObjectByQueryObj` — search object

## Description

```
array hw_getobjectbyqueryobj (int connection, string query, int max_hits);
```

Searches for objects on the whole server and returns an array of object records. The maximum number of matches is limited to *max\_hits*. If *max\_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also `hw_GetObjectByQuery`.

## hw\_GetObjectByQueryColl

### Name

`hw_GetObjectByQueryColl` — search object in collection

### Description

```
array hw_getobjectbyquerycoll (int connection, int objectID, string query, int max_hits);
```

Searches for objects in collection with ID *objectID* and returns an array of object ids. The maximum number of matches is limited to *max\_hits*. If *max\_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also `hw_GetObjectByQueryCollObj`.

## hw\_GetObjectByQueryCollObj

### Name

hw\_GetObjectByQueryCollObj — search object in collection

### Description

```
array hw_getobjectbyquerycollobj (int connection, int objectID, string query,  
int max_hits);
```

Searches for objects in collection with ID *objectID* and returns an array of object records. The maximum number of matches is limited to *max\_hits*. If *max\_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also hw\_GetObjectByQueryColl.

## hw\_GetChildDocColl

### Name

hw\_GetChildDocColl — object ids of child documents of collection

### Description

```
array hw_getchilddoccoll (int connection, int objectID);
```

Returns array of object ids for child documents of a collection.

See also hw\_GetChildren, hw\_GetChildColl.

## hw\_GetChildDocCollObj

### Name

hw\_GetChildDocCollObj — object records of child documents of collection

### Description

```
array hw_getchilddoccollobj(int connection, int objectID);
```

Returns an array of object records for child documents of a collection.

See also hw\_ChildrenObj, hw\_GetChildCollObj.

## hw\_GetAnchors

### Name

hw\_GetAnchors — object ids of anchors of document

### Description

```
array hw_getanchors(int connection, int objectID);
```

Returns an array of object ids with anchors of the document with object ID *objectID*.

## hw\_GetAnchorsObj

### Name

hw\_GetAnchorsObj — object records of anchors of document

### Description

```
array hw_getanchorsobj(int connection, int objectID);
```

Returns an array of object records with anchors of the document with object ID *objectID*.

## hw\_Mv

### Name

hw\_Mv — moves objects

### Description

```
int hw_mv(int connection, array object id array, int source id, int
destination id);
```

Moves the objects with object ids as specified in the second parameter from the collection with id *source id* to the collection with the id *destination id*. If the destination id is 0 the objects will be unlinked from the source collection. If this is the last instance of that object it will be deleted. If you want to delete all instances at once, use `hw_deleteobject`.

The value return is the number of moved objects.

See also `hw_cp`, `hw_deleteobject`.

## hw\_Identify

### Name

`hw_Identify` — identifies as user

### Description

```
int hw_identify(string username, string password);
```

Identifies as user with *username* and *password*. Identification is only valid for the current session. I do not think this function will be needed very often. In most cases it will be easier to identify with the opening of the connection.

See also `hw_Connect`.

## hw\_InCollections

### Name

`hw_InCollections` — check if object ids in collections

### Description

```
array hw_incollections(int connection, array object_id_array, array  
collection_id_array, int return_collections);
```

Checks whether a set of objects (documents or collections) specified by the *object\_id\_array* is part of the collections listed in *collection\_id\_array*. When the fourth parameter *return\_collections* is 0, the subset of object ids that is part of the collections (i.e., the documents or collections that are children of one or more collections of collection ids or their subcollections, recursively) is returned as an array. When the fourth parameter is 1, however, the set of collections that have one or more objects of this subset as children are returned as an array. This option allows a client to,

e.g., highlight the part of the collection hierarchy that contains the matches of a previous query, in a graphical overview.

## hw\_Info

### Name

hw\_Info — info about connection

### Description

```
string hw_info(int connection);
```

Returns information about the current connection. The returned string has the following format:  
<Serverstring>, <Host>, <Port>, <Username>, <Port of Client>, <Byte swapping>

## hw\_InsColl

### Name

hw\_InsColl — insert collection

### Description

```
int hw_inscoll(int connection, int objectID, array object_array);
```

Inserts a new collection with attributes as in *object\_array* into collection with object ID *objectID*.

## hw\_InsDoc

### Name

hw\_InsDoc — insert document

### Description

```
int hw_insdDoc(int connection, int parentID, string object_record, string
text);
```

Inserts a new document with attributes as in *object\_record* into collection with object ID *parentID*. This function inserts either an object record only or an object record and a pure ascii text in *text* if *text* is given. If you want to insert a general document of any kind use `hw_insertdocument` instead.

See also `hw_InsertDocument`, `hw_InsColl`.

## hw\_InsertDocument

### Name

hw\_InsertDocument — upload any document

### Description

```
int hw_insertdocument(int connection, int parent_id, int hw_document);
```

Uploads a document into the collection with *parent\_id*. The document has to be created before with `hw_NewDocument`. Make sure that the object record of the new document contains at least the attributes: Type, DocumentType, Title and Name. Possibly you also want to set the MimeType. The functions returns the object id of the new document or false.

See also `hw_PipeDocument`.

## hw\_InsertObject

### Name

hw\_InsertObject — inserts an object record

### Description

```
int hw_insertobject(int connection, string object rec, string parameter);
```

Inserts an object into the server. The object can be any valid hyperwave object. See the HG-CSP documentation for a detailed information on how the parameters have to be.

Note: If you want to insert an Anchor, the attribute Position has always been set either to a start/end value or to 'invisible'. Invisible positions are needed if the annotation has no correspondig link in the annotation text.

See also hw\_PipeDocument, hw\_InsertDocument, hw\_InsDoc, hw\_InsColl.

## hw\_mapid

### Name

hw\_mapid — Maps global id on virtual local id

### Description

```
int hw_mapid(int connection, int server id, int object id);
```

Maps a global object id on any hyperwave server, even those you did not connect to with hw\_connect, onto a virtual object id. This virtual object id can then be used as any other object id, e.g. to obtain the object record with hw\_getobject. The server id is the first part of the global object id (GOid) of the object which is actually the IP number as an integer.

Note: In order to use this function you will have to set the `F_DISTRIBUTED` flag, which can currently only be set at compile time in `hg_comm.c`. It is not set by default. Read the comment at the beginning of `hg_comm.c`

## hw\_Modifyobject

### Name

`hw_Modifyobject` — modifies object record

### Description

```
int hw_modifyobject (int connection, int object_to_change, array remove, array
add, int mode);
```

This command allows to remove, add, or modify individual attributes of an object record. The object is specified by the Object ID *object\_to\_change*. The first array *remove* is a list of attributes to remove. The second array *add* is a list of attributes to add. In order to modify an attribute one will have to remove the old one and add a new one. `hw_modifyobject` will always remove the attributes before it adds attributes unless the value of the attribute to remove is not a string or array.

The last parameter determines if the modification is performed recursively. 1 means recursive modification. If some of the objects cannot be modified they will be skipped without notice. `hw_error` may not indicate an error though some of the objects could not be modified.

The keys of both arrays are the attributes name. The value of each array element can either be an array, a string or anything else. If it is an array each attribute value is constructed by the key of each element plus a colon and the value of each element. If it is a string it is taken as the attribute value. An empty string will result in a complete removal of that attribute. If the value is neither a string nor an array but something else, e.g. an integer, no operation at all will be performed on the attribute. This is necessary if you want to add a completely new attribute not just a new value for an existing attribute. If the remove array contained an empty string for that attribute, the attribute would be tried to be removed which would fail since it doesn't exist. The following addition of a new value for that attribute would also fail. Setting the value for that attribute to e.g. 0 would not even try to remove it and the addition will work.

If you would like to change the attribute 'Name' with the current value 'books' into 'articles' you will have to create two arrays and call `hw_modifyobject`.

**Example 1. modifying an attribute**

```
// $connect is an existing connection to the Hyperwave server
// $objid is the ID of the object to modify
$remarr = array("Name" => "books");
$addarr = array("Name" => "articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

In order to delete/add a name=value pair from/to the object record just pass the remove/add array and set the last/third parameter to an empty array. If the attribute is the first one with that name to add, set attribute value in the remove array to an integer.

**Example 2. adding a completely new attribute**

```
// $connect is an existing connection to the Hyperwave server
// $objid is the ID of the object to modify
$remarr = array("Name" => 0);
$addarr = array("Name" => "articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

**Note:** Multilingual attributes, e.g. 'Title', can be modified in two ways. Either by providing the attributes value in its native form 'language': 'title' or by providing an array with elements for each language as described above. The above example would than be:

**Example 3. modifying Title attribute**

```
$remarr = array("Title" => "en:Books");
$addarr = array("Title" => "en:Articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

or

**Example 4. modifying Title attribute**

```
$remarr = array("Title" => array("en" => "Books"));
$addarr = array("Title" => array("en" => "Articles", "ge"=>"Artikel"));
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

This removes the english title 'Books' and adds the english title 'Articles' and the german title 'Artikel'.

#### Example 5. removing attribute

```
$remarr = array("Title" => "");
$addarr = array("Title" => "en:Articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

**Note:** This will remove all attributes with the name 'Title' and adds a new 'Title' attribute. This comes in handy if you want to remove attributes recursively.

**Note:** If you need to delete all attributes with a certain name you will have to pass an empty string as the attribute value.

**Note:** Only the attributes 'Title', 'Description' and 'Keyword' will properly handle the language prefix. If those attributes don't carry a language prefix, the prefix 'xx' will be assigned.

**Note:** The 'Name' attribute is somewhat special. In some cases it cannot be complete removed. You will get an error message 'Change of base attribute' (not clear when this happens). Therefore you will always have to add a new Name first and then remove the old one.

**Note:** You may not surround this function by calls to `hw_getandlock` and `hw_unlock`. `hw_modifyobject` does this internally.

Returns TRUE if no error occurs otherwise FALSE.

## hw\_New\_Document

### Name

hw\_New\_Document — create new document

### Description

```
int hw_new_document (string object_record, string document_data, int
document_size);
```

Returns a new Hyperwave document with document data set to *document\_data* and object record set to *object\_record*. The length of the *document\_data* has to passed in *document\_size*. This function does not insert the document into the Hyperwave server.

See also hw\_FreeDocument, hw\_DocumentSize, hw\_DocumentBodyTag, hw\_OutputDocument, hw\_InsertDocument.

## hw\_Objrec2Array

### Name

hw\_Objrec2Array — convert attributes from object record to object array

### Description

```
array hw_objrec2array (string object_record);
```

Converts an *object\_record* into an object array. The keys of the resulting array are the attributes names. Multiple attributes like 'Title' in different languages form its own array. The keys of this array are the left part to the colon of the attribute value. Currently only the attributes 'Title', 'Description' and 'Keyword' are treated properly. Other multiple attributes form an index array. Currently only the attribute 'Group' is handled properly.

See also `hw_array2objrec`.

## hw\_OutputDocument

### Name

`hw_OutputDocument` — prints `hw_document`

### Description

```
int hw_outputdocument (int hw_document);
```

Prints the document without the BODY tag.

## hw\_pConnect

### Name

`hw_pConnect` — make a persistent database connection

### Description

```
int hw_pconnect (string host, int port, string username, string password);
```

Returns a connection index on success, or false if the connection could not be made. Opens a persistent connection to a Hyperwave server. Each of the arguments should be a quoted string, except for the port number. The *username* and *password* arguments are optional and can be left out. In such a case no identification with the server will be done. It is similar to identify as user anonymous. This function returns a connection index that is needed by other Hyperwave functions. You can have multiple persistent connections open at once.

See also `hw_Connect`.

## hw\_PipeDocument

### Name

`hw_PipeDocument` — retrieve any document

### Description

```
int hw_pipedocument (int connection, int objectID);
```

Returns the Hyperwave document with object ID *objectID*. If the document has anchors which can be inserted, they will have been inserted already. The document will be transferred via a special data connection which does not block the control connection.

See also `hw_GetText` for more on link insertion, `hw_FreeDocument`, `hw_DocumentSize`, `hw_DocumentBodyTag`, `hw_OutputDocument`.

## hw\_Root

### Name

`hw_Root` — root object id

### Description

```
int hw_root();
```

Returns the object ID of the hyperroot collection. Currently this is always 0. The child collection of the hyperroot is the root collection of the connected server.

## hw\_Unlock

### Name

hw\_Unlock — unlock object

### Description

```
int hw_unlock(int connection, int objectID);
```

Unlocks a document, so other users regain access.

See also hw\_GetAndLock.

## hw\_Who

### Name

hw\_Who — List of currently logged in users

### Description

```
int hw_who(int connection);
```

Returns an array of users currently logged into the Hyperwave server. Each entry in this array is an array itself containing the elements id, name, system, onSinceDate, onSinceTime, TotalTime and self. 'self' is 1 if this entry belongs to the user who initiated the request.

## hw\_Username

### Name

hw\_Username — name of currently logged in user

### Description

```
string hw_getusername(int connection);
```

Returns the username of the connection.

# XXI. Image functions

You can use the image functions in PHP to get the size of JPEG, GIF, and PNG images, and if you have the GD library (available at <http://www.boutell.com/gd/>) you will also be able to create and manipulate images.

# GetImageSize

## Name

GetImageSize — get the size of a GIF, JPG or PNG image

## Description

```
array getimagesize(string filename, array [imageinfo]);
```

The GetImageSize function will determine the size of any GIF, JPG or PNG image file and return the dimensions along with the file type and a height/width text string to be used inside a normal HTML IMG tag.

Returns an array with 4 elements. Index 0 contains the width of the image in pixels. Index 1 contains the height. Index 2 a flag indicating the type of the image. 1 = GIF, 2 = JPG, 3 = PNG. Index 3 is a text string with the correct "height=xxx width=xxx" string that can be used directly in an IMG tag.

### Example 1. GetImageSize

```
<?php $size = GetImageSize("img/flag.jpg"); ?>
<IMG SRC="img/flag.jpg" <?php echo $size[3]; ?>
```

The optional *imageinfo* parameter allows you to extract some extended information from the image file. Currently this will return the different JPG APP markers in an associative Array. Some Programs use these APP markers to embed text information in images. A very common one is to embed IPTC <http://www.xe.net/iptc/> information in the APP13 marker. You can use the *iptcparse* function to parse the binary APP13 marker into something readable.

### Example 2. GetImageSize returning IPTC

```
<?php
    $size = GetImageSize("testimg.jpg",&$info);
    if (isset($info["APP13"])) {
        $iptc = iptcparse($info["APP13"]);
        var_dump($iptc);
    }
?>
```

**Note:** This function does not require the GD image library.

## ImageArc

### Name

ImageArc — draw a partial ellipse

### Description

```
int imagearc(int im, int cx, int cy, int w, int h, int s, int e, int col);
```

ImageArc draws a partial ellipse centered at *cx*, *cy* (top left is 0,0) in the image represented by *im*. *w* and *h* specifies the ellipse's width and height respectively while the start and end points are specified in degrees indicated by the *s* and *e* arguments.

## ImageChar

### Name

ImageChar — draw a character horizontally

### Description

```
int imagechar(int im, int font, int x, int y, string c, int col);
```

ImageChar draws the first character of *c* in the image identified by *id* with its upper-left at *x,y* (top left is 0,0) with the color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used (with higher numbers corresponding to larger fonts).

See also `imagemloadfont`.

## ImageCharUp

### Name

`ImageCharUp` — draw a character vertically

### Description

```
int imagecharup(int im, int font, int x, int y, string c, int col);
```

`ImageCharUp` draws the character *c* vertically in the image identified by *im* at coordinates *x*, *y* (top left is 0, 0) with the color *col*. If *font* is 1, 2, 3, 4 or 5, a built-in font is used.

See also `imagemloadfont`.

## ImageColorAllocate

### Name

`ImageColorAllocate` — allocate a color for an image

### Description

```
int imagecolorallocate(int im, int red, int green, int blue);
```

`ImageColorAllocate` returns a color identifier representing the color composed of the given RGB components. The *im* argument is the return from the `imagecreate` function. `ImageColorAllocate` must be called to create each color that is to be used in the image represented by *im*.

```
$white = ImageColorAllocate($im, 255,255,255);  
$black = ImageColorAllocate($im, 0,0,0);
```

## ImageColorAt

### Name

`ImageColorAt` — get the index of the color of a pixel

### Description

```
int imagecolorat(int im, int x, int y);
```

Returns the index of the color of the pixel at the specified location in the image.

See also `imagecolorset` and `imagecolorsforindex`.

## ImageColorClosest

### Name

`ImageColorClosest` — get the index of the closest color to the specified color

### Description

```
int imagecolorclosest(int im, int red, int green, int blue);
```

Returns the index of the color in the palette of the image which is "closest" to the specified RGB value.

The "distance" between the desired color and each color in the palette is calculated as if the RGB values represented points in three-dimensional space.

See also `imagecolorexact`.

## ImageColorExact

### Name

`ImageColorExact` — get the index of the specified color

### Description

```
int imagecolorexact(int im, int red, int green, int blue);
```

Returns the index of the specified color in the palette of the image.

If the color does not exist in the image's palette, -1 is returned.

See also `imagecolorclosest`.

## ImageColorResolve

### Name

`ImageColorResolve` — get the index of the specified color or its closest possible alternative

### Description

```
int imagecolorresolve(int im, int red, int green, int blue);
```

This function is guaranteed to return a color index for a requested color, either the exact color or the closest possible alternative.

See also `imagecolorclosest`.

## ImageColorSet

### Name

`ImageColorSet` — set the color for the specified palette index

### Description

```
bool imagecolorset(int im, int index, int red, int green, int blue);
```

This sets the specified index in the palette to the specified color. This is useful for creating flood-fill-like effects in paletted images without the overhead of performing the actual flood-fill.

See also `imagecolorat`.

## ImageColorsForIndex

### Name

`ImageColorsForIndex` — get the colors for an index

### Description

```
array imagecolorsforindex(int im, int index);
```

This returns an associative array with red, green, and blue keys that contain the appropriate values for the specified color index.

See also `imagecolorat` and `imagecolorexact`.

## ImageColorsTotal

### Name

`ImageColorsTotal` — find out the number of colors in an image's palette

### Description

```
int imagecolorstotal (int im);
```

This returns the number of colors in the specified image's palette.

See also `imagecolorat` and `imagecolorsforindex`.

## ImageColorTransparent

### Name

`ImageColorTransparent` — define a color as transparent

### Description

```
int imagecolortransparent (int im, int [col]);
```

`ImageColorTransparent` sets the transparent color in the `im` image to `col`. `im` is the image identifier returned by `imagecreate` and `col` is a color identifier returned by `imagecolorallocate`.

The identifier of the new (or current, if none is specified) transparent color is returned.

## ImageCopyResized

### Name

ImageCopyResized — copy and resize part of an image

### Description

```
int imagecopyresized(int dst_im, int src_im, int dstX, int dstY, int srcX,  
int srcY, int dstW, int dstH, int srcW, int srcH);
```

ImageCopyResized copies a rectangular portion of one image to another image. *dst\_im* is the destination image, *src\_im* is the source image identifier. If the source and destination coordinates and width and heights differ, appropriate stretching or shrinking of the image fragment will be performed. The coordinates refer to the upper left corner. This function can be used to copy regions within the same image (if *dst\_im* is the same as *src\_im*) but if the regions overlap the results will be unpredictable.

## ImageCreate

### Name

ImageCreate — create a new image

### Description

```
int imagecreate(int x_size, int y_size);
```

ImageCreate returns an image identifier representing a blank image of size *x\_size* by *y\_size*.

# ImageCreateFromGif

## Name

ImageCreateFromGif — create a new image from file or URL

## Description

```
int imagecreatefromgif(string filename);
```

imagecreatefromgif returns an image identifier representing the image obtained from the given filename.

imagecreatefromgif returns an empty string on failure. It also outputs an error message, which unfortunately displays as a broken link in a browser. To ease debugging the following example will produce an error GIF:

### Example 1. Example to handle an error during creation (courtesy vic@zysys.com )

```
function LoadGif($imgname)
{
    $im = @imagecreatefromgif($imgname); /* Attempt to open */
    if ($im == "") { /* See if it failed */
        $im = ImageCreate(150,30); /* Create a blank image */
        $bgc = ImageColorAllocate($im,255,255,255);
        $tc = ImageColorAllocate($im,0,0,0);
        ImageFilledRectangle($im,0,0,150,30,$bgc);
        ImageString($im,1,5,5,"Error loading $imgname",$tc); /* Output an er-
rmmsg */
    }
    return $im;
}
```

**Note:** Since all GIF support was removed from the GD library in version 1.6, this function is not available if you are using that version of the GD library.

## ImageDashedLine

### Name

ImageDashedLine — draw a dashed line

### Description

```
int imagedashedline(int im, int x1, int y1, int x2, int y2, int col);
```

ImageLine draws a dashed line from x1,y1 to x2,y2 (top left is 0,0) in image im of color col.

See also `imageline`.

## ImageDestroy

### Name

ImageDestroy — destroy an image

### Description

```
int imagedestroy(int im);
```

ImageDestroy frees any memory associated with image im. im is the image identifier returned by the `imagecreate` function.

# ImageFill

## Name

ImageFill — flood fill

## Description

```
int imagefill(int im, int x, int y, int col);
```

ImageFill performs a flood fill starting at coordinate x, y (top left is 0,0) with color col in the image im.

# ImageFilledPolygon

## Name

ImageFilledPolygon — draw a filled polygon

## Description

```
int imagefilledpolygon(int im, array points, int num_points, int col);
```

ImageFilledPolygon creates a filled polygon in image im. points is a PHP array containing the polygon's vertices, ie. points[0] = x0, points[1] = y0, points[2] = x1, points[3] = y1, etc. num\_points is the total number of vertices.

## ImageFilledRectangle

### Name

ImageFilledRectangle — draw a filled rectangle

### Description

```
int imagefilledrectangle(int im, int x1, int y1, int x2, int y2, int col);
```

ImageFilledRectangle creates a filled rectangle of color *col* in image *im* starting at upper left coordinates *x1*, *y1* and ending at bottom right coordinates *x2*, *y2*. 0, 0 is the top left corner of the image.

## ImageFillToBorder

### Name

ImageFillToBorder — flood fill to specific color

### Description

```
int imagefilltoborder(int im, int x, int y, int border, int col);
```

ImageFillToBorder performs a flood fill whose border color is defined by *border*. The starting point for the fill is *x*,*y* (top left is 0,0) and the region is filled with color *col*.

## ImageFontHeight

### Name

ImageFontHeight — get font height

### Description

```
int imagefontheight(int font);
```

Returns the pixel height of a character in the specified font.

See also `imagefontwidth` and `imageloadfont`.

## ImageFontWidth

### Name

ImageFontWidth — get font width

### Description

```
int imagefontwidth(int font);
```

Returns the pixel width of a character in font.

See also `imagefontheight` and `imageloadfont`.

# ImageGif

## Name

ImageGif — output image to browser or file

## Description

```
int imagegif(int im, string filename);
```

`imagegif` creates the GIF file in `filename` from the image `im`. The `im` argument is the return from the `imagecreate` function.

The image format will be GIF87a unless the image has been made transparent with `imagecolortransparent`, in which case the image format will be GIF89a.

The filename argument is optional, and if left off, the raw image stream will be output directly. By sending an image/gif content-type using `header`, you can create a PHP script that outputs GIF images directly.

**Note:** Since all GIF support was removed from the GD library in version 1.6, this function is not available if you are using that version of the GD library.

# ImageInterlace

## Name

ImageInterlace — enable or disable interlace

## Description

```
int imageinterlace(int im, int [interlace]);
```

`ImageInterlace` turns the interlace bit on or off. If interlace is 1 the image will be interlaced, and if interlace is 0 the interlace bit is turned off.

This function returns whether the interlace bit is set for the image.

## ImageLine

### Name

`ImageLine` — draw a line

### Description

```
int imageline(int im, int x1, int y1, int x2, int y2, int col);
```

`ImageLine` draws a line from *x1*,*y1* to *x2*,*y2* (top left is 0,0) in image *im* of color *col*.

See also `imagecreate` and `imagecolorallocate`.

## ImageLoadFont

### Name

`ImageLoadFont` — load a new font

### Description

```
int imagerloadfont(string file);
```

`ImageLoadFont` loads a user-defined bitmap font and returns an identifier for the font (that is always greater than 5, so it will not conflict with the built-in fonts).

The font file format is currently binary and architecture dependent. This means you should generate the font files on the same type of CPU as the machine you are running PHP on.

**Table 1. Font file format**

byte position	C data type	description
byte 0-3	int	number of characters in the font
byte 4-7	int	value of first character in the font (often 32 for space)
byte 8-11	int	pixel width of each character
byte 12-15	int	pixel height of each character
byte 16-	char	array with character data, one byte per pixel in each character, for a total of (nchars*width*height) bytes.

See also `ImageFontWidth` and `ImageFontHeight`.

## ImagePolygon

### Name

`ImagePolygon` — draw a polygon

### Description

```
int imagepolygon(int im, array points, int num_points, int col);
```

`ImagePolygon` creates a polygon in image `id`. `points` is a PHP array containing the polygon's vertices, ie. `points[0] = x0`, `points[1] = y0`, `points[2] = x1`, `points[3] = y1`, etc. `num_points` is the total number of vertices.

See also `imagecreate`.

# ImagePSBBox

## Name

ImagePSBBox — give the bounding box of a text rectangle using PostScript Type1 fonts

## Description

```
array imagepsbbox(string text, int font, int size, int space, int width,  
float angle);
```

*size* is expressed in pixels.

*space* allows you to change the default value of a space in a font. This amount is added to the normal value and can also be negative.

*tightness* allows you to control the amount of white space between characters. This amount is added to the normal character width and can also be negative.

*angle* is in degrees.

Parameters *space* and *tightness* are expressed in character space units, where 1 unit is 1/1000th of an em-square.

Parameters *space*, *tightness* and *angle* are optional.

The bounding box is calculated using information available from character metrics, and unfortunately tends to differ slightly from the results achieved by actually rasterizing the text. If the angle is 0 degrees, you can expect the text to need 1 pixel more to every direction.

This function returns an array containing the following elements:

0	lower left x-coordinate
1	lower left y-coordinate
2	upper right x-coordinate
3	upper right y-coordinate

See also `imagepstext`.

## ImagePSEncodeFont

### Name

ImagePSEncodeFont — change the character encoding vector of a font

### Description

```
int imagepsencodefont (string encodingfile);
```

Loads a character encoding vector from a file and changes the fonts encoding vector to it. As a PostScript fonts default vector lacks most of the character positions above 127, you'll definitely want to change this if you use an other language than english. The exact format of this file is described in T1libs documentation. T1lib comes with two ready-to-use files, IsoLatin1.enc and IsoLatin2.enc.

If you find yourself using this function all the time, a much better way to define the encoding is to set `ps.default_encoding` in the configuration file to point to the right encoding file and all fonts you load will automatically have the right encoding.

## ImagePSFreeFont

### Name

ImagePSFreeFont — free memory used by a PostScript Type 1 font

### Description

```
void imagepsfreefont (int fontindex);
```

See also `imagepsloadfont`.

## ImagePSLoadFont

### Name

ImagePSLoadFont — load a PostScript Type 1 font from file

### Description

```
int imagepsloadfont(string filename);
```

In the case everything went right, a valid font index will be returned and can be used for further purposes. Otherwise the function returns false and prints a message describing what went wrong.

See also `imagepsfreefont`.

## ImagePSText

### Name

ImagePSText — to draw a text string over an image using PostScript Type1 fonts

### Description

```
array imagepstext(int image, string text, int font, int size, int foreground,  
int background, int x, int y, int [space], int [tightness], float [angle],  
int [antialias_steps]);
```

*size* is expressed in pixels.

*foreground* is the color in which the text will be painted. *background* is the color to which the text will try to fade in with antialiasing. No pixels with the color *background* are actually painted, so the background image does not need to be of solid color.

The coordinates given by *x*, *y* will define the origin (or reference point) of the first character (roughly the lower-left corner of the character). This is different from the `ImageString`, where *x*, *y* define the upper-right corner of the first character. Refer to PostScript documentation about fonts and their measuring system if you have trouble understanding how this works.

*space* allows you to change the default value of a space in a font. This amount is added to the normal value and can also be negative.

*tightness* allows you to control the amount of white space between characters. This amount is added to the normal character width and can also be negative.

*angle* is in degrees.

*antialias\_steps* allows you to control the number of colours used for antialiasing text. Allowed values are 4 and 16. The higher value is recommended for text sizes lower than 20, where the effect in text quality is quite visible. With bigger sizes, use 4. It's less computationally intensive.

Parameters *space* and *tightness* are expressed in character space units, where 1 unit is 1/1000th of an em-square.

Parameters *space*, *tightness*, *angle* and *antialias* are optional.

This function returns an array containing the following elements:

0	lower left x-coordinate
1	lower left y-coordinate
2	upper right x-coordinate
3	upper right y-coordinate

See also `imagepsbbox`.

## ImageRectangle

### Name

`ImageRectangle` — draw a rectangle

## Description

```
int imagerectangle(int im, int x1, int y1, int x2, int y2, int col);
```

ImageRectangle creates a rectangle of color *col* in image *im* starting at upper left coordinate *x1*,*y1* and ending at bottom right coordinate *x2*,*y2*. 0,0 is the top left corner of the image.

## ImageSetPixel

### Name

ImageSetPixel — set a single pixel

### Description

```
int imagesetpixel(int im, int x, int y, int col);
```

ImageSetPixel draws a pixel at *x*,*y* (top left is 0,0) in image *im* of color *col*.

See also `imagecreate` and `imagecolorallocate`.

## ImageString

### Name

ImageString — draw a string horizontally

### Description

```
int imagestring(int im, int font, int x, int y, string s, int col);
```

ImageString draws the string *s* in the image identified by *im* at coordinates *x,y* (top left is 0,0) in color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used.

See also `imagedloadfont`.

## ImageStringUp

### Name

ImageStringUp — draw a string vertically

### Description

```
int imagestringup(int im, int font, int x, int y, string s, int col);
```

ImageStringUp draws the string *s* vertically in the image identified by *im* at coordinates *x,y* (top left is 0,0) in color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used.

See also `imagedloadfont`.

## ImageSX

### Name

ImageSX — get image width

### Description

```
int imagesx(int im);
```

ImageSX returns the width of the image identified by *im*.

See also `imagecreate` and `imagesy`.

## ImageSY

### Name

ImageSY — get image height

### Description

```
int imagesy(int im);
```

ImageSY returns the height of the image identified by `im`.

See also `imagecreate` and `imagesx`.

## ImageTTFBBox

### Name

ImageTTFBBox — give the bounding box of a text using TrueType fonts

### Description

```
array ImageTTFBBox(int size, int angle, string fontfile, string text);
```

This function calculates and returns the bounding box in pixels a TrueType text.

*text*

The string to be measured.

*size*

The font size.

*fontfile*

The name of the TrueType font file. (Can also be an URL.)

*angle*

Angle in degrees in which *text* will be measured.

ImageTTFBBox returns an array with 8 elements representing four points making the bounding box of the text:

0	lower left corner, X position
1	lower left corner, Y position
2	lower right corner, X position
3	lower right corner, Y position
4	upper right corner, X position
5	upper right corner, Y position
6	upper left corner, X position
7	upper left corner, Y position

The points are relative to the *text* regardless of the angle, so "upper left" means in the top left-hand corner seeing the text horizontally.

This function requires both the GD library and the Freetype library.

See also ImageTTFText.

# ImageTTFText

## Name

ImageTTFText — write text to the image using TrueType fonts

## Description

```
array ImageTTFText (int im, int size, int angle, int x, int y, int col, string
fontfile, string text);
```

ImageTTFText draws the string *text* in the image identified by *im*, starting at coordinates *x,y* (top left is 0,0), at an angle of *angle* in color *col*, using the TrueType font file identified by *fontfile*.

The coordinates given by *x,y* will define the basepoint of the first character (roughly the lower-left corner of the character). This is different from the ImageString, where *x,y* define the upper-right corner of the first character.

*angle* is in degrees, with 0 degrees being left-to-right reading text (3 o'clock direction), and higher values representing a counter-clockwise rotation. (i.e., a value of 90 would result in bottom-to-top reading text).

*fontfile* is the path to the TrueType font you wish to use.

*text* is the text string which may include UTF-8 character sequences (of the form: `&#123;`) to access characters in a font beyond the first 255.

*col* is the color index. Using the negative of a color index has the effect of turning off antialiasing.

ImageTTFText returns an array with 8 elements representing four points making the bounding box of the text. The order of the points is upper left, upper right, lower right, lower left. The points are relative to the text regardless of the angle, so "upper left" means in the top left-hand corner when you see the text horizontally.

This example script will produce a black GIF 400x30 pixels, with the words "Testing..." in white in the font Arial.

### Example 1. ImageTTFText

```
<?php
Header("Content-type: image/gif");
$im = imagecreate(400,30);
```

```
$black = ImageColorAllocate($im, 0,0,0);  
$white = ImageColorAllocate($im, 255,255,255);  
ImageTTFText($im, 20, 0, 10, 20, $white, "/path/arial.ttf", "Test-  
ing... Omega: &#937;");  
ImageGif($im);  
ImageDestroy($im);  
?>
```

This function requires both the GD library and the FreeType (<http://www.freetype.org/>) library.

See also ImageTTFBBox.

## XXII. IMAP functions

To get these functions to work, you have to compile PHP with `-with-imap`. That requires the `c-client` library to be installed. Grab the latest version from <ftp://ftp.cac.washington.edu/imap/> and compile it. Then copy `c-client/c-client.a` to `/usr/local/lib` or some other directory on your link path and copy `c-client/rfc822.h`, `mail.h` and `linkage.h` to `/usr/local/include` or some other directory in your include path.

# imap\_append

## Name

imap\_append — Append a string message to a specified mailbox

## Description

```
int imap_append(int imap_stream, string mbbox, string message, string flags);
```

Returns true on success, false on error.

imap\_append appends a string message to the specified mailbox *mbbox*. If the optional *flags* is specified, writes the *flags* to that mailbox also.

When talking to the Cyrus IMAP server, you must use "\r\n" as your end-of-line terminator instead of "\n" or the operation will fail.

# imap\_base64

## Name

imap\_base64 — Decode BASE64 encoded text

## Description

```
string imap_base64(string text);
```

imap\_base64 function decodes BASE-64 encoded text. The decoded message is returned as a string.

## imap\_body

### Name

`imap_body` — Read the message body

### Description

```
string imap_body(int imap_stream, int msg_number, int flags);
```

`imap_body` returns the body of the message, numbered *msg\_number* in the current mailbox. The optional *flags* are a bit mask with one or more of the following:

- FT\_UID - The msgno is a UID
- FT\_PEEK - Do not set the \Seen flag if not already set
- FT\_INTERNAL - The return string is in internal format, will not canonicalize to CRLF.

## imap\_check

### Name

`imap_check` — Check current mailbox

### Description

```
array imap_check(int imap_stream);
```

Returns information about the current mailbox. Returns FALSE on failure.

The `imap_check` function checks the current mailbox status on the server and returns the information in an object with following properties.

Date : date of the message  
Driver : driver  
Mailbox : name of the mailbox  
Nmsgs : number of messages  
Recent : number of recent messages

## imap\_close

### Name

`imap_close` — Close an IMAP stream

### Description

```
int imap_close(int imap_stream, int flags);
```

Close the imap stream. Takes an optional *flag* CL\_EXPUNGE, which will silently expunge the mailbox before closing.

## imap\_createmailbox

### Name

`imap_createmailbox` — Create a new mailbox

### Description

```
int imap_createmailbox(int imap_stream, string mbox);
```

`imap_createmailbox` creates a new mailbox specified by *mbbox*.

Returns true on success and false on error.

## imap\_delete

### Name

`imap_delete` — Mark a message for deletion from current mailbox

### Description

```
int imap_delete(int imap_stream, int msg_number);
```

Returns true.

`imap_delete` function marks message pointed by *msg\_number* for deletion. Actual deletion of the messages is done by `imap_expunge`.

## imap\_deletemailbox

### Name

`imap_deletemailbox` — Delete a mailbox

### Description

```
int imap_deletemailbox(int imap_stream, string mbbox);
```

`imap_deletemailbox` deletes the specified mailbox.

Returns true on success and false on error.

## imap\_expunge

### Name

`imap_expunge` — Delete all messages marked for deletion

### Description

```
int imap_expunge(int imap_stream);
```

`imap_expunge` deletes all the messages marked for deletion by `imap_delete`.

Returns true.

## imap\_fetchbody

### Name

`imap_fetchbody` — Fetch a particular section of the body of the message

### Description

```
string imap_fetchbody(int imap_stream, int msg_number, string part_number,  
flags flags);
```

This function causes a fetch of a particular section of the body of the specified messages as a text string and returns that text string. The section specification is a string of integers delimited by period which index into a body part list as per the IMAP4 specification. Body parts are not decoded by this function.

The options for `imap_fetchbody` are a bitmask with one or more of the following

- FT\_UID - The msgno is a UID
- FT\_PEEK - Do not set the \Seen flag if not already set

- FT\_UID - The return string is in "internal" format, without any attempt to canonicalize CRLF

## imap\_fetchstructure

### Name

`imap_fetchstructure` — Read the structure of a particular message

### Description

```
array imap_fetchstructure (int imap_stream, int msg_number);
```

This function causes a fetch of all the structured information for the given *msg\_number*. The returned value is an object with following elements.

type, encoding, ifsubtype, subtype, ifdescription, description, ifid,  
id, lines, bytes, ifparameters

It also returns an array of objects called `parameters[]`. This object has following properties.

attribute, value

In case of multipart, it also returns an array of objects of all the properties, called `parts[]`.

# imap\_header

## Name

`imap_header` — Read the header of the message

## Description

```
object imap_header(int imap_stream, int msg_number, int fromlength, int
subjectlength, int defaulthost);
```

This function returns an object of various header elements

*remail*,*date*,*Date*,*subject*,*Subject*,*in\_reply\_to*,*message\_id*,*newsgroups*,  
    *followup\_to*,*references*

message flags:

    Recent - 'R' if recent and seen, 'N' if recent and not seen, ' ' if not recent  
    Unseen - 'U' if not seen AND not recent, ' ' if seen OR not seen and recent  
    Answered - 'A' if answered, ' ' if unanswered  
    Deleted - 'D' if deleted, ' ' if not deleted  
    Draft - 'X' if draft, ' ' if not draft  
    Flagged - 'F' if flagged, ' ' if not flagged

NOTE that the Recent/Unseen behavior is a little odd. If you want to know if a message is Unseen, you must check for

Unseen == 'U' || Recent == 'N'

*toaddress* (full to: line, up to 1024 characters)

*to*[] (returns an array of objects from the To line, containing:)

*personal*  
    *adl*  
    *mailbox*  
    *host*

*fromaddress* (full from: line, up to 1024 characters)

from[] (returns an array of objects from the From line, containing:)

- personal
- adl
- mailbox
- host

ccaddress (full cc: line, up to 1024 characters)

cc[] (returns an array of objects from the Cc line, containing:)

- personal
- adl
- mailbox
- host

bccaddress (full bcc line, up to 1024 characters)

bcc[] (returns an array of objects from the Bcc line, containing:)

- personal
- adl
- mailbox
- host

reply\_toaddress (full reply\_to: line, up to 1024 characters)

reply\_to[] (returns an array of objects from the Reply\_to line, containing:)

- personal
- adl
- mailbox
- host

senderaddress (full sender: line, up to 1024 characters)

sender[] (returns an array of objects from the sender line, containing:)

- personal
- adl
- mailbox
- host

return\_path (full return-path: line, up to 1024 characters)

return\_path[] (returns an array of objects from the return\_path line, containing:)

- personal
- adl
- mailbox
- host

update ( mail message date in unix time)

fetchfrom (from line formatted to fit *fromlength* characters)

fetchsubject (subject line formatted to fit *subjectlength* characters)

## imap\_headers

### Name

imap\_headers — Returns headers for all messages in a mailbox

### Description

```
array imap_headers (int imap_stream);
```

Returns an array of string formatted with header info. One element per mail message.

## imap\_listmailbox

### Name

imap\_listmailbox — Read the list of mailboxes

### Description

```
array imap_listmailbox (int imap_stream, string ref, string pat);
```

Returns an array containing the names of the mailboxes.

## imap\_getmailboxes

### Name

`imap_getmailboxes` — Read the list of mailboxes, returning detailed information on each one

### Description

```
array imap_getmailboxes (int imap_stream, string ref, string pat);
```

Returns an array of objects containing mailbox information. Each object has the attributes *name*, specifying the full name of the mailbox; *delimiter*, which is the hierarchy delimiter for the part of the hierarchy this mailbox is in; and *attributes*. *Attributes* is a bitmask that can be tested against:

- LATT\_NOINFERIORS - This mailbox has no "children" (there are no mailboxes below this one)
- LATT\_NOSELECT - This is only a container, not a mailbox - you cannot open it.
- LATT\_MARKED - This mailbox is marked. Only used by UW-IMAPD.
- LATT\_UNMARKED - This mailbox is not marked. Only used by UW-IMAPD.

*ref* should normally be just the IMAP server, in the form: {imap\_server:imap\_port}, and *pattern* specifies where in the mailbox hierarchy to start searching. If you want all mailboxes, pass *pattern* as an empty string.

There are two special characters you can pass as part of the *pattern*: '\*' and '%'. '\*' means to return all mailboxes. If you pass *pattern* as '\*', you will get a list of the entire mailbox hierarchy. '%' means to return the current level only. '%' as the *pattern* parameter will return only the top level mailboxes; '~/' on UW-IMAPD will return every mailbox in the ~/mail directory, but none in subfolders of that directory.

## imap\_listsubscribed

### Name

imap\_listsubscribed — List all the subscribed mailboxes

### Description

```
array imap_listsubscribed(int imap_stream, string ref, string pattern);
```

Returns an array of all the mailboxes that you have subscribed. The *ref* and *pattern* arguments specify the base location to search from and the pattern the mailbox name must match.

## imap\_getsubscribed

### Name

imap\_getsubscribed — List all the subscribed mailboxes

### Description

```
array imap_getsubscribed(int imap_stream, string ref, string pattern);
```

This function is identical to `imap_getmailboxes`, except that it only returns mailboxes that the user is subscribed to.

## imap\_mail\_copy

### Name

`imap_mail_copy` — Copy specified messages to a mailbox

### Description

```
int imap_mail_copy(int imap_stream, string msglist, string mbox, int flags);
```

Returns true on success and false on error.

Copies mail messages specified by *msglist* to specified mailbox. *msglist* is a range not just message numbers.

*flags* is a bitmask of one or more of

- CP\_UID - the sequence numbers contain UIDS
- CP\_MOVE - Delete the messages from the current mailbox after copying

## imap\_mail\_move

### Name

`imap_mail_move` — Move specified messages to a mailbox

### Description

```
int imap_mail_move(int imap_stream, string msglist, string mbox);
```

Moves mail messages specified by *msglist* to specified mailbox. *msglist* is a range not just message numbers.

Returns true on success and false on error.

## imap\_num\_msg

### Name

`imap_num_msg` — Gives the number of messages in the current mailbox

### Description

```
int imap_num_msg(int stream_id);
```

Return the number of messages in the current mailbox.

## imap\_num\_recent

### Name

`imap_num_recent` — Gives the number of recent messages in current mailbox

### Description

```
int imap_num_recent(int imap_stream);
```

Returns the number of recent messages in the current mailbox.

# imap\_open

## Name

`imap_open` — Open an IMAP stream to a mailbox

## Description

```
int imap_open(string mailbox, string username, string password, int flags);
```

Returns an IMAP stream on success and false on error. This function can also be used to open streams to POP3 and NNTP servers. To connect to an IMAP server running on port 143 on the local machine, do the following:

```
$mbox = imap_open("{localhost:143}INBOX", "user_id", "password");
```

To connect to a POP3 server on port 110 on the local server, use:

```
$mbox = imap_open("{localhost/pop3:110}INBOX", "user_id", "password");
```

To connect to an NNTP server on port 119 on the local server, use:

```
$nntp = imap_open("{localhost/nntp:119}comp.test", "", "");
```

To connect to a remote server replace "localhost" with the name or the IP address of the server you want to connect to.

The options are a bit mask with one or more of the following:

- `OP_READONLY` - Open mailbox read-only
- `OP_ANONYMOUS` - Dont use or update a .newsrsrc for news
- `OP_HALFOPEN` - For IMAP and NNTP names, open a connection but dont open a mailbox
- `CL_EXPUNGE` - Expunge mailbox automatically upon mailbox close

## imap\_ping

### Name

`imap_ping` — Check if the IMAP stream is still active

### Description

```
int imap_ping(int imap_stream);
```

Returns true if the stream is still alive, false otherwise.

`imap_ping` function pings the stream to see it is still active. It may discover new mail; this is the preferred method for a periodic "new mail check" as well as a "keep alive" for servers which have inactivity timeout.

## imap\_renamemailbox

### Name

`imap_renamemailbox` — Rename an old mailbox to new mailbox

### Description

```
int imap_renamemailbox(int imap_stream, string old_mbox, string new_mbox);
```

This function renames on old mailbox to new mailbox.

Returns true on success and false on error.

## imap\_reopen

### Name

`imap_reopen` — Reopen IMAP stream to new mailbox

### Description

```
int imap_reopen(string imap_stream, string mailbox, string [flags]);
```

Returns true on success and false on error.

This function reopens the specified stream to new mailbox.

The options are a bit mask with one or more of the following:

- `OP_READONLY` - Open mailbox read-only
- `OP_ANONYMOUS` - Dont use or update a .newsrsrc for news
- `OP_HALFOPEN` - For IMAP and NNTP names, open a connection but dont open a mailbox
- `CL_EXPUNGE` - Expunge mailbox automatically upon mailbox close

## imap\_subscribe

### Name

`imap_subscribe` — Subscribe to a mailbox

### Description

```
int imap_subscribe(int imap_stream, string mbox);
```

Subscribe to a new mailbox.

Returns true on success and false on error.

## imap\_undelete

### Name

`imap_undelete` — Unmark the message which is marked deleted

### Description

```
int imap_undelete(int imap_stream, int msg_number);
```

This function removes the deletion flag for a specified message, which is set by `imap_delete`.

Returns true on success and false on error.

## imap\_unsubscribe

### Name

`imap_unsubscribe` — Unsubscribe from a mailbox

### Description

```
int imap_unsubscribe(int imap_stream, string mbx);
```

Unsubscribe from a specified mailbox.

Returns true on success and false on error.

## imap\_qprint

### Name

`imap_qprint` — Convert a quoted-printable string to an 8 bit string

### Description

```
string imap_qprint(string string);
```

Convert a quoted-printable string to an 8 bit string

Returns an 8 bit (binary) string

## imap\_8bit

### Name

`imap_8bit` — Convert an 8bit string to a quoted-printable string.

### Description

```
string imap_8bit(string string);
```

Convert an 8bit string to a quoted-printable string.

Returns a quoted-printable string

## imap\_binary

### Name

`imap_binary` — Convert an 8bit string to a base64 string.

### Description

```
string imap_binary(string string);
```

Convert an 8bit string to a base64 string.

Returns a base64 string

## imap\_scanmailbox

### Name

`imap_scanmailbox` — Read the list of mailboxes, takes a string to search for in the text of the mailbox

### Description

```
array imap_scanmailbox(int imap_stream, string string);
```

Returns an array containing the names of the mailboxes that have *string* in the text of the mailbox.

## imap\_mailboxmsginfo

### Name

`imap_mailboxmsginfo` — Get information about the current mailbox

### Description

```
array imap_mailboxmsginfo(int imap_stream);
```

Returns information about the current mailbox. Returns FALSE on failure.

The `imap_mailboxmsginfo` function checks the current mailbox status on the server and returns the information in an object with following properties.

- Date : date of the message
- Driver : driver
- Mailbox : name of the mailbox
- Nmsgs : number of messages
- Recent : number of recent messages
- Unread : number of unread messages
- Size : mailbox size

## imap\_rfc822\_write\_address

### Name

`imap_rfc822_write_address` — Returns a properly formatted email address given the mailbox, host, and personal info.

## Description

```
string imap_rfc822_write_address (string mailbox, string host, string personal);
```

Returns a properly formatted email address given the mailbox, host, and personal info.

## imap\_rfc822\_parse\_adrlist

### Name

`imap_rfc822_parse_adrlist` — Parses an address string

### Description

```
string imap_rfc822_parse_adrlist (string address, string default_host);
```

This function parses the address string and for each address, returns an array of objects. The 4 objects are:

- mailbox - the mailbox name (username)
- host - the host name
- personal - the personal name
- adl - at domain source route

## imap\_setflag\_full

### Name

imap\_setflag\_full — Sets flags on messages

### Description

```
string imap_setflag_full(int stream, string sequence, string flag, string  
options);
```

This function causes a store to add the specified flag to the flags set for the messages in the specified sequence.

The options are a bit mask with one or more of the following:

ST_UID	The sequence argument contains UIDs instead of sequence numbers
--------	-----------------------------------------------------------------

## imap\_clearflag\_full

### Name

imap\_clearflag\_full — Clears flags on messages

### Description

```
string imap_clearflag_full(int stream, string sequence, string flag, string  
options);
```

This function causes a store to delete the specified flag to the flags set for the messages in the specified sequence.

The options are a bit mask with one or more of the following:

ST\_UID        The sequence argument contains UIDs instead of  
sequence numbers

## imap\_sort

### Name

imap\_sort —

### Description

```
string imap_sort(int stream, int criteria, int reverse, int options);
```

Returns an array of message numbers sorted by the given parameters

Rev is 1 for reverse-sorting.

Criteria can be one (and only one) of the following:

SORTDATE	message Date
SORTARRIVAL	arrival date
SORTFROM	mailbox in first From address
SORTSUBJECT	message Subject
SORTTO	mailbox in first To address
SORTCC	mailbox in first cc address
SORTSIZE	size of message in octets

The flags are a bitmask of one or more of the following:

SE\_UID        Return UIDs instead of sequence numbers  
 SE\_NOPREFETCH   Don't prefetch searched messages.

## imap\_fetchheader

### Name

`imap_fetchheader` — Returns header for a message

### Description

```
string imap_fetchheader(int imap_stream, int msgno, int flags);
```

This function causes a fetch of the complete, unfiltered RFC 822 format header of the specified message as a text string and returns that text string.

The options are:

FT\_UID        The `msgno` argument is a UID  
 FT\_INTERNAL   The return string is in "internal" format,  
               without any attempt to canonicalize to CRLF  
               newlines  
 FT\_PREFETCHTEXT The RFC822.TEXT should be pre-fetched at the  
                   same time. This avoids an extra RTT on an  
                   IMAP connection if a full message text is  
                   desired (e.g. in a "save to local file"  
                   operation)

## imap\_uid

### Name

`imap_uid` — This function returns the UID for the given message sequence number.

### Description

```
int imap_uid(int imap_stream, int msgno);
```

This function returns the UID for the given message sequence number. It is the inverse of `imap_msgno`.

## imap\_msgno

### Name

`imap_msgno` — This function returns the message sequence number for the given UID.

### Description

```
int imap_msgno(int imap_stream, int uid);
```

This function returns the message sequence number for the given UID. It is the inverse of `imap_uid`.

## imap\_search

### Name

`imap_search` — This function returns an array of messages matching the given search criteria.

### Description

```
array imap_search(int imap_stream, string criteria, int flags);
```

This function performs a search on the mailbox currently opened in the given imap stream. *criteria* is a string, delimited by spaces, in which the following keywords are allowed. Any multi-word arguments (eg FROM "joe smith") must be quoted.

- ALL - return all messages matching the rest of the criteria
- ANSWERED - match messages with the \ANSWERED flag set
- BCC "string" - match messages with "string" in the Bcc: field
- BEFORE "date" - match messages with Date: before "date"
- BODY "string" - match messages with "string" in the body of the message
- CC "string" - match messages with "string" in the Cc: field
- DELETED - match deleted messages
- FLAGGED - match messages with the \FLAGGED (sometimes referred to as Important or Urgent) flag set
- FROM "string" - match messages with "string" in the From: field
- KEYWORD "string" - match messages with "string" as a keyword
- NEW - match new messages
- OLD - match old messages
- ON "date" - match messages with Date: matching "date"
- RECENT - match messages with the \RECENT flag set
- SEEN - match messages that have been read (the \SEEN flag is set)
- SINCE "date" - match messages with Date: after "date"
- SUBJECT "string" - match messages with "string" in the Subject:

- TEXT "string" - match messages with text "string"
- TO "string" - match messages with "string" in the To:
- UNANSWERED - match messages that have not been answered
- UNDELETED - match messages that are not deleted
- UNFLAGGED - match messages that are not flagged
- UNKEYWORD "string" - match messages that do not have the keyword "string"
- UNSEEN - match messages which have not been read yet

For example, to match all unanswered messages sent by Mom, you'd use: "UNANSWERED FROM mom". Searches appear to be case insensitive. This list of criteria is from a reading of the UW c-client source code and may be uncomplete or inaccurate. Searcher beware.

Valid values for flags are SE\_UID, which causes the returned array to contain UIDs instead of messages sequence numbers.

## imap\_last\_error

### Name

`imap_last_error` — This function returns the last IMAP error (if any) that occurred during this page request.

### Description

```
string imap_last_error(void );
```

This function returns the full text of the last IMAP error message that occurred on the current page. The error stack is untouched; calling `imap_last_error` subsequently, with no intervening errors, will return the same error.

## imap\_errors

### Name

`imap_errors` — This function returns all of the IMAP errors (if any) that have occurred during this page request or since the error stack was reset.

### Description

```
array imap_errors(void );
```

This function returns an array of all of the IMAP error messages generated since the last `imap_errors` call, or the beginning of the page. When `imap_errors` is called, the error stack is subsequently cleared.

## imap\_alerts

### Name

`imap_alerts` — This function returns all IMAP alert messages (if any) that have occurred during this page request or since the alert stack was reset.

### Description

```
array imap_alerts(void );
```

This function returns an array of all of the IMAP alert messages generated since the last `imap_alerts` call, or the beginning of the page. When `imap_alerts` is called, the alert stack is subsequently cleared. The IMAP specification requires that these messages be passed to the user.

# imap\_status

## Name

`imap_status` — This function returns status information on a mailbox other than the current one.

## Description

```
object imap_status(int imap_stream, string mailbox, int options);
```

This function returns an object containing status information. Valid flags are:

- `SA_MESSAGES` - set `status->messages` to the number of messages in the mailbox
- `SA_RECENT` - set `status->recent` to the number of recent messages in the mailbox
- `SA_UNSEEN` - set `status->unseen` to the number of unseen (new) messages in the mailbox
- `SA_UIDNEXT` - set `status->uidnext` to the next uid to be used in the mailbox
- `SA_UIDVALIDITY` - set `status->uidvalidity` to a constant that changes when uids for the mailbox may no longer be valid
- `SA_ALL` - set all of the above

`status->flags` is also set, which contains a bitmask which can be checked against any of the above constants.

## **XXIII. PHP options & information**

# error\_log

## Name

error\_log — send an error message somewhere

## Description

```
int error_log(string message, int message_type, string [destination], string [extra_headers]);
```

Sends an error message to the web server's error log, a TCP port or to a file. The first parameter, *message*, is the error message that should be logged. The second parameter, *message\_type* says where the message should go:

**Table 1. error\_log log types**

0	<i>message is sent to PHP's system logger, using the Operating System's system logging mechanism or a file, depending on what the error_log configuration directive is set to.</i>
1	<i>message is sent by email to the address in the destination parameter. This is the only message type where the fourth parameter, extra_headers is used. This message type uses the same internal function as Mail does.</i>

2	message is sent through the PHP debugging connection. This option is only available if remote debugging has been enabled. In this case, the destination parameter specifies the host name or IP address and optionally, port number, of the socket receiving the debug information.
3	message is appended to the file destination.

**Example 1. error\_log examples**

```
// Send notification through the server log if we can not
// connect to the database.
if (!Ora_Logon($username, $password)) {
    error_log("Oracle database not available!", 0);
}

// Notify administrator by email if we run out of FOO
if (!($foo = allocate_new_foo())) {
    error_log("Big trouble, we're all out of FOOs!", 1,
        "operator@mydomain.com");
}

// other ways of calling error_log():
error_log("You messed up!", 2, "127.0.0.1:7000");
error_log("You messed up!", 2, "loghost");
error_log("You messed up!", 3, "/var/tmp/my-errors.log");
```

## error\_reporting

### Name

`error_reporting` — set which PHP errors are reported

### Description

```
int error_reporting(int [level]);
```

Sets PHP's error reporting level and returns the old level. The error reporting level is a bitmask of the following values (follow the links for the internal values to get their meanings):

**Table 1. `error_reporting` bit values**

value	internal name
1	<code>E_ERROR</code>
2	<code>E_WARNING</code>
4	<code>E_PARSE</code>
8	<code>E_NOTICE</code>
16	<code>E_CORE_ERROR</code>
32	<code>E_CORE_WARNING</code>

## extension\_loaded

### Name

`extension_loaded` — find out whether an extension is loaded

## Description

```
bool extension_loaded(string name);
```

Returns true if the extension identified by *name* is loaded. You can see the names of various extensions by using `phpinfo`.

See also `phpinfo`.

**Note:** This function was added in 3.0.10.

## getenv

### Name

`getenv` — Get the value of an environment variable

### Description

```
string getenv(string varname);
```

Returns the value of the environment variable *varname*, or false on an error.

```
$ip = getenv("REMOTE_ADDR"); // get the ip number of the user
```

You can see a list of all the environmental variables by using `phpinfo`. You can find out what many of them mean by taking a look at the CGI specification (<http://hoohoo.ncsa.uiuc.edu/cgi/>), specifically the page on environmental variables (<http://hoohoo.ncsa.uiuc.edu/cgi/env.html>).

## get\_cfg\_var

### Name

`get_cfg_var` — Get the value of a PHP configuration option.

### Description

```
string get_cfg_var(string varname);
```

Returns the current value of the PHP configuration variable specified by *varname*, or false if an error occurs.

It will not return configuration information set when the PHP was compiled, or read from an Apache configuration file (using the `php3_configuration_option` directives).

To check whether the system is using a configuration file, try retrieving the value of the `cfg_file_path` configuration setting. If this is available, a configuration file is being used.

## get\_current\_user

### Name

`get_current_user` — Get the name of the owner of the current PHP script.

### Description

```
string get_current_user(void);
```

Returns the name of the owner of the current PHP script.

See also `getmyuid`, `getmypid`, `getmyinode`, and `getlastmod`.

## get\_magic\_quotes\_gpc

### Name

`get_magic_quotes_gpc` — Get the current active configuration setting of magic quotes gpc.

### Description

```
long get_magic_quotes_gpc (void);
```

Returns the current active configuration setting of `magic_quotes_gpc`. (0 for off, 1 for on)

See also `get_magic_quotes_runtime`, `set_magic_quotes_runtime`.

## get\_magic\_quotes\_runtime

### Name

`get_magic_quotes_runtime` — Get the current active configuration setting of `magic_quotes_runtime`.

### Description

```
long get_magic_quotes_runtime (void);
```

Returns the current active configuration setting of `magic_quotes_runtime`. (0 for off, 1 for on)

See also `get_magic_quotes_gpc`, `set_magic_quotes_runtime`.

# getlastmod

## Name

getlastmod — Get time of last page modification.

## Description

```
int getlastmod(void);
```

Returns the time of the last modification of the current page. The value returned is a Unix timestamp, suitable for feeding to `date`. Returns false on error.

### Example 1. `getlastmod()` example

```
// outputs e.g. 'Last modified: March 04 1998 20:43:59.'  
echo "Last modified: ".date( "F d Y H:i:s.", getlastmod() );
```

See also `date`, `getmyuid`, `get_current_user`, `getmyinode`, and `getmypid`.

# getmyinode

## Name

getmyinode — Get the inode of the current script.

## Description

```
int getmyinode(void);
```

Returns the current script's inode, or false on error.

See also `getmyuid`, `get_current_user`, `getmypid`, and `getlastmod`.

## getmypid

### Name

`getmypid` — Get PHP's process ID.

### Description

```
int getmypid(void);
```

Returns the current PHP process ID, or false on error.

Note that when running as a server module, separate invocations of the script are not guaranteed to have distinct pids.

See also `getmyuid`, `get_current_user`, `getmyinode`, and `getlastmod`.

## getmyuid

### Name

`getmyuid` — Get PHP script owner's UID.

### Description

```
int getmyuid(void);
```

Returns the user ID of the current script, or false on error.

See also `getmypid`, `get_current_user`, `getmyinode`, and `getlastmod`.

# getrusage

## Name

getrusage — Get the current resource usages.

## Description

```
array getrusage(int [who]);
```

This is an interface to getrusage(2). It returns an associative array containing the data returned from the system call. If who is 1, getrusage will be called with RUSAGE\_CHILDREN. All entries are accessible by using their documented field names.

### Example 1. Getrusage Example

```
$dat = getrusage();
echo $dat["ru_nswap"];           # number of swaps
echo $dat["ru_majflt"];          # number of page faults
echo $dat["ru_utime.tv_sec"];    # user time used (seconds)
echo $dat["ru_utime.tv_usec"];  # user time used (microseconds)
```

See your system's man page for more details.

# phpinfo

## Name

phpinfo — Output lots of PHP information.

## Description

```
int phpinfo(void);
```

Outputs a large amount of information about the current state of PHP. This includes information about PHP compilation options and extensions, the PHP version, server information and environment (if compiled as a module), the PHP environment, OS version information, paths, master and local values of configuration options, HTTP headers, and the GNU Public License.

See also `phpversion`.

## phpversion

### Name

`phpversion` — Get the current PHP version.

### Description

```
string phpversion(void);
```

Returns a string containing the version of the currently running PHP parser.

#### Example 1. `phpversion()` example

```
// prints e.g. 'Current PHP version: 3.0rel-dev'  
echo "Current PHP version: ".phpversion();
```

See also `phpinfo`.

## putenv

### Name

`putenv` — Set the value of an environment variable.

## Description

```
void putenv(string setting);
```

Adds *setting* to the environment.

### Example 1. Setting an Environment Variable

```
putenv("UNIQID=$uniqid");
```

## set\_magic\_quotes\_runtime

### Name

`set_magic_quotes_runtime` — Set the current active configuration setting of `magic_quotes_runtime`.

### Description

```
long set_magic_quotes_runtime(int new_setting);
```

Set the current active configuration setting of `magic_quotes_runtime`. (0 for off, 1 for on)

See also `get_magic_quotes_gpc`, `get_magic_quotes_runtime`.

# set\_time\_limit

## Name

`set_time_limit` — limit the maximum execution time

## Description

```
void set_time_limit(int seconds);
```

Set the number of seconds a script is allowed to run. If this is reached, the script returns a fatal error. The default limit is 30 seconds or, if it exists, the `max_execution_time` value defined in the configuration file. If `seconds` is set to zero, no time limit is imposed.

When called, `set_time_limit` restarts the timeout counter from zero. In other words, if the timeout is the default 30 seconds, and 25 seconds into script execution a call such as `set_time_limit(20)` is made, the script will run for a total of 45 seconds before timing out.

## XXIV. Informix functions

The Informix driver for Online (ODS) 7.x, SE 7.x and Universal Server (IUS) 9.x is implemented in "functions/ifx.ec" and "functions/php3\_ifx.h". ODS 7.x support is fairly complete, with full support for BYTE and TEXT columns. IUS 9.x support is partly finished: the new data types are there, but SLOB and CLOB support is still under construction.

### Configuration notes:

Before you run the "configure" script, make sure that the "INFORMIXDIR" variable has been set.

The configure script will autdetect the libraries and include directories, if you run "configure --with\_informix=yes". You can override this detection by specifying "IFX\_LIBDIR", "IFX\_LIBS" and "IFX\_INCDIR" in the environment. The configure script will also try to detect your Informix server version. It will set the "HAVE\_IFX\_IUS" conditional compilation variable if your Informix version >= 9.00.

### Some notes on the use of BLOBs (TEXT and BYTE columns):

BLOBs are normally addressed by integer BLOB identifiers. Select queries return a "blob id" for every BYTE and TEXT column. You can get at the contents with "string\_var = ifx\_get\_blob(\$blob\_id);" if you choose to get the BLOBs in memory (with : "ifx\_blobinfile(0);"). If you prefer to receive the content of BLOB columns in a file, use "ifx\_blobinfile(1);", and "ifx\_get\_blob(\$blob\_id);" will get you the filename. Use normal file I/O to get at the blob contents.

For insert/update queries you must create these "blob id's" yourself with "ifx\_create\_blob(..)". You then plug the blob id's into an array, and replace the blob columns with a question mark (?) in the query string. For updates/inserts, you are responsible for setting the blob contents with ifx\_update\_blob(...).

The behaviour of BLOB columns can be altered by configuration variables that also can be set at runtime :

configuration variable : ifx.textasvarchar

configuration variable : ifx.byteasvarchar

runtime functions :

ifx\_textasvarchar(0) : use blob id's for select queries with TEXT columns

ifx\_byteasvarchar(0) : use blob id's for select queries with BYTE columns

ifx\_textasvarchar(1) : return TEXT columns as if they were VARCHAR columns, without the use of blob id's for select queries.

ifx\_byteasvarchar(1) : return BYTE columns as if they were VARCHAR columns, without the use of blob id's for select queries.

configuration variable : ifx.blobinfile

runtime function :

ifx\_blobinfile\_mode(0) : return BYTE columns in memory, the blob id lets you get at the contents.

ifx\_blobinfile\_mode(1) : return BYTE columns in a file, the blob id lets you get at the file name.

If you set ifx\_text/byteasvarchar to 1, you can use TEXT and BYTE columns in select queries just like normal (but rather long) VARCHAR fields. Since all strings are "counted" in PHP, this remains "binary safe". It is up to you to handle this correctly. The returned data can contain anything, you are responsible for the contents.

If you set ifx\_blobinfile to 1, use the file name returned by ifx\_get\_blob(..) to get at the blob contents. Note that in this case YOU ARE RESPONSIBLE FOR DELETING THE TEMPORARY FILES CREATED BY INFORMIX when fetching the row. Every new row fetched will create new temporary files for every BYTE column.

The location of the temporary files can be influenced by the environment variable "blobdir", default is "." (the current directory). Something like : putenv(blobdir=tmpblob"); will ease the cleaning up of temp files accidentally left behind (their names all start with "blb").

#### **Automatically trimming "char" (SQLCHAR and SQLNCHAR) data:**

This can be set with the configuration variable

ifx.charasvarchar : if set to 1 trailing spaces will be automatically trimmed.

#### **NULL values:**

The configuration variable ifx.nullformat (and the runtime function ifx\_nullformat) when set to true will return NULL columns as the string "NULL", when set to false they return the empty string. This allows you to discriminate between NULL columns and empty columns.

## ifx\_connect

### Name

`ifx_connect` — Open Informix server connection

### Description

```
int ifx_connect (string [database] , string [userid] , string [password] );
```

Returns an connection identifier on success, or FALSE on error.

`ifx_connect` establishes a connection to an Informix server. All of the arguments are optional, and if they're missing, defaults are taken from values supplied in configuration file (`ifx.default_host` for the host (Informix libraries will use `INFORMIXSERVER` environment value if not defined), `ifx.default_user` for user, `ifx.default_password` for the password (none if not defined).

In case a second call is made to `ifx_connect` with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned.

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `ifx_close`.

See also `ifx_pconnect`, and `ifx_close`.

#### Example 1. Connect to a Informix database

```
$conn_id = ifx_pconnect (mydb@ol_srv1, "imyself", "mypassword");
```

## ifx\_pconnect

### Name

`ifx_pconnect` — Open persistent Informix connection

## Description

```
int ifx_pconnect (string [database] , string [userid] , string [password] );
```

Returns: A positive Informix persistent link identifier on success, or false on error

`ifx_pconnect` acts very much like `ifx_connect` with two major differences.

This function behaves exactly like `ifx_connect` when PHP is not running as an Apache module. First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`ifx_close` will not close links established by `ifx_pconnect`).

This type of links is therefore called 'persistent'.

See also: `ifx_connect`.

## ifx\_close

### Name

`ifx_close` — Close Informix connection

### Description

```
int ifx_close(int [link_identifier] );
```

Returns: always true.

`ifx_close` closes the link to an Informix database that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note that this isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

`ifx_close` will not close persistent links generated by `ifx_pconnect`.

See also: `ifx_connect`, and `ifx_pconnect`.

### Example 1. Closing a Informix connection

```
$conn_id = ifx_connect (mydb@ol_srv, "itsme", "mypassword");
... some queries and stuff ...
ifx_close($conn_id);
```

## ifx\_query

### Name

`ifx_query` — Send Informix query

### Description

```
int ifx_query(string query, int [link_identifier] , int [cursor_type] , mixed
[blobidarray] );
```

Returns: A positive Informix result identifier on success, or false on error.

An integer "result\_id" used by other functions to retrieve the query results. Sets "affected\_rows" for retrieval by the `ifx_affected_rows` function.

`ifx_query` sends a query to the currently active database on the server that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed. If no link is open, the function tries to establish a link as if `ifx_connect` was called, and use it.

Executes *query* on connection *conn\_id*. For "select-type" queries a cursor is declared and opened. The optional *cursor\_type* parameter allows you to make this a "scroll" and/or "hold" cursor. It's a mask and can be either `IFX_SCROLL`, `IFX_HOLD`, or both or'ed together. Non-select queries are "execute immediate".

For either query type the number of (estimated or real) affected rows is saved for retrieval by `ifx_affected_rows`.

If you have BLOB (BYTE or TEXT) columns in an update query, you can add a *blobidarray* parameter containing the corresponding "blob ids", and you should replace those columns with a "?" in the query text.

If the contents of the TEXT (or BYTE) column allow it, you can also use "ifx\_textasvarchar(1)" and "ifx\_byteasvarchar(1)". This allows you to treat TEXT (or BYTE) columns just as if they were ordinary (but long) VARCHAR columns for select queries, and you don't need to bother with blob id's.

With ifx\_textasvarchar(0) or ifx\_byteasvarchar(0) (the default situation), select queries will return BLOB columns as blob id's (integer value). You can get the value of the blob as a string or file with the blob functions (see below).

See also: ifx\_connect.

### Example 1. Show all rows of the "orders" table as a html table

```
ifx_textasvarchar(1);          // use "text mode" for blobs
$res_id = ifx_query("select * from orders", $conn_id);
if (! $res_id) {
    printf("Can't select orders : %s\n<br>%s<br>\n", ifx_error());
    ifx_errormsg();
    die;
}
ifx_htmltbl_result($res_id, "border=\"1\"");
ifx_free_result($res_id);
```

### Example 2. Insert some values into the "catalog" table

```
                // create blob id's for a byte and text column
$textid = ifx_create_blob(0, 0, "Text column in memory");
$byteid = ifx_create_blob(1, 0, "Byte column in memory");
                // store blob id's in a blobid array
$blobidarray[] = $textid;
$blobidarray[] = $byteid;
                // launch query
$query = "insert into catalog (stock_num, manu_code, " .
        "cat_descr,cat_picture) values(1,'HRO',?,?,?)";
$res_id = ifx_query($query, $conn_id, $blobidarray);
if (! $res_id) {
    ... error ...
}
                // free result id
ifx_free_result($res_id);
```

## ifx\_prepare

### Name

`ifx_prepare` — Prepare an SQL-statement for execution

### Description

```
int ifx_prepare(string query, int conn_id, int [cursor_def], mixed
blobidarray);
```

Returns a integer *result\_id* for use by `ifx_do`. Sets *affected\_rows* for retrieval by the `ifx_affected_rows` function.

Prepares *query* on connection *conn\_id*. For "select-type" queries a cursor is declared and opened. The optional *cursor\_type* parameter allows you to make this a "scroll" and/or "hold" cursor. It's a mask and can be either `IFX_SCROLL`, `IFX_HOLD`, or both or'ed together.

For either query type the estimated number of affected rows is saved for retrieval by `ifx_affected_rows`.

If you have BLOB (BYTE or TEXT) columns in the query, you can add a *blobidarray* parameter containing the corresponding "blob ids", and you should replace those columns with a "?" in the query text.

If the contents of the TEXT (or BYTE) column allow it, you can also use `"ifx_textasvarchar(1)"` and `"ifx_byteasvarchar(1)"`. This allows you to treat TEXT (or BYTE) columns just as if they were ordinary (but long) VARCHAR columns for select queries, and you don't need to bother with blob id's.

With `ifx_textasvarchar(0)` or `ifx_byteasvarchar(0)` (the default situation), select queries will return BLOB columns as blob id's (integer value). You can get the value of the blob as a string or file with the blob functions (see below).

See also: `ifx_do`.

## ifx\_do

### Name

`ifx_do` — Execute a previously prepared SQL-statement

### Description

```
int ifx_do(int result_id);
```

Returns TRUE on success, FALSE on error.

Executes a previously prepared query or opens a cursor for it.

Does NOT free *result\_id* on error.

Also sets the real number of `ifx_affected_rows` for non-select statements for retrieval by `ifx_affected_rows`

See also: `ifx_prepare`. There is a example.

## ifx\_error

### Name

`ifx_error` — Returns error code of last Informix call

### Description

```
string ifx_error(void);
```

The Informix error codes (SQLSTATE & SQLCODE) formatted as follows :

x [SQLSTATE = aa bbb SQLCODE=cccc]

where x = space : no error

E : error

N : no more data

W : warning

? : undefined

If the "x" character is anything other than space, SQLSTATE and SQLCODE describe the error in more detail.

See the Informix manual for the description of SQLSTATE and SQLCODE

Returns in a string one character describing the general results of a statement and both SQLSTATE and SQLCODE associated with the most recent SQL statement executed. The format of the string is "(char) [SQLSTATE=(two digits) (three digits) SQLCODE=(one digit)]". The first character can be ' ' (space) (success), 'w' (the statement caused some warning), 'E' (an error happened when executing the statement) or 'N' (the statement didn't return any data).

See also: `ifx_errormsg`

## ifx\_errormsg

### Name

`ifx_errormsg` — Returns error message of last Informix call

### Description

```
string ifx_errormsg(int [errorcode]);
```

Returns the Informix error message associated with the most recent Informix error, or, when the optional "errorcode" param is present, the error message corresponding to "errorcode".

See also: `ifx_error`

```
printf("%s\n<br>", ifx_errormsg(-201));
```

## ifx\_affected\_rows

### Name

`ifx_affected_rows` — Get number of rows affected by a query

### Description

```
int ifx_affected_rows (int result_id);
```

*result\_id* is a valid result id returned by `ifx_query` or `ifx_prepare`.

Returns the number of rows affected by a query associated with *result\_id*.

For inserts, updates and deletes the number is the real number (`sqlerrd[2]`) of affected rows. For selects it is an estimate (`sqlerrd[0]`). Don't rely on it.

Useful after `ifx_prepare` to limit queries to reasonable result sets.

See also: `ifx_num_rows`

#### Example 1. Informix affected rows

```
$rid = ifx_prepare ("select * from emp where name like " . $name, $connid);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
```

## ifx\_getsqlca

### Name

`ifx_getsqlca` — Get the contents of `sqlca.sqlerrd[0..5]` after a query

### Description

```
array ifx_getsqlca (int result_id);
```

*result\_id* is a valid result id returned by `ifx_query` or `ifx_prepare`.

Returns a pseudo-row (associative array) with `sqlca.sqlerrd[0]` to `sqlca.sqlerrd[5]` after the query associated with *result\_id*.

For inserts, updates and deletes the values returned are those as set by the server after executing the query. This gives access to the number of affected rows and the serial insert value. For selects the values are those saved after the prepare statement. This gives access to the estimated number of affected rows. The use of this function saves the overhead of executing a "select dbinfo('sqlca.sqlerrdx')" query, as it retrieves the values that were saved by the ifx driver at the appropriate moment.

#### Example 1. Retrieve Informix `sqlca.sqlerrd[x]` values

```
/* assume the first column of 'sometable' is a serial */
$qid = ifx_query("insert into sometable values(0, '2nd column', 'another col-
umn' ", $connid);
if (! $qid) {
    ... error ...
}
$sqlca = ifx_getsqlca ($qid);
$serial_value = $sqlca["sqlerrd1"];
echo "The serial value of the inserted row is : " . $serial_value<br>\n";
```

## ifx\_fetch\_row

### Name

`ifx_fetch_row` — Get row as enumerated array

### Description

```
array ifx_fetch_row(int result_id, mixed [position] );
```

Returns an associative array that corresponds to the fetched row, or false if there are no more rows.

Blob columns are returned as integer blob id values for use in `ifx_get_blob` unless you have used `ifx_textasvarchar(1)` or `ifx_byteasvarchar(1)`, in which case blobs are returned as string values. Returns FALSE on error

*result\_id* is a valid resultid returned by `ifx_query` or `ifx_prepare` (select type queries only!).

[*position*] is an optional parameter for a "fetch" operation on "scroll" cursors: "NEXT", "PREVIOUS", "CURRENT", "FIRST", "LAST" or a number. If you specify a number, an "absolute" row fetch is executed. This parameter is optional, and only valid for scrollcursors.

`ifx_fetch_row` fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to `ifx_fetch_row` would return the next row in the result set, or false if there are no more rows.

#### Example 1. Informix fetch rows

```
$rid = ifx_prepare ("select * from emp where name like " . $name,
                  $connid, IFX_SCROLL);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
if (! ifx_do ($rid)) {
    ... error ...
}
```

```

}
$row = ifx_fetch_row ($rid, "NEXT");
while (is_array($row)) {
    for(reset($row); $fieldname=key($row); next($row)) {
        $fieldvalue = $row[$fieldname];
        printf ("%s = %s,", $fieldname, $fieldvalue);
    }
    printf("\n<br>");
    $row = ifx_fetch_row ($rid, "NEXT");
}
ifx_free_result ($rid);

```

## ifx\_htmltbl\_result

### Name

`ifx_htmltbl_result` — Formats all rows of a query into a HTML table

### Description

```
int ifx_htmltbl_result(int result_id, string [html_table_options]);
```

Returns the number of rows fetched or FALSE on error.

Formats all rows of the *result\_id* query into a html table. The optional second argument is a string of <table> tag options

#### Example 1. Informix results as HTML table

```

$rid = ifx_prepare ("select * from emp where name like " . $name,
                  $connid, IFX_SCROLL);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {

```

```

        printf ("Too many rows in result set (%d)\n<br>", $rowcount);
        die ("Please restrict your query<br>\n");
    }
    if (! ifx_do($rid) {
        ... error ...
    }

    ifx_htmltbl_result ($rid, "border=\"2\"");

    ifx_free_result($rid);

```

## ifx\_fieldtypes

### Name

`ifx_fieldtypes` — List of Informix SQL fields

### Description

array **ifx\_fieldtypes** (int *result\_id*);

Returns an associative array with fieldnames as key and the SQL fieldtypes as data for query with *result\_id*. Returns FALSE on error.

#### Example 1. Fieldnames and SQL fieldtypes

```

$types = ifx_fieldtypes ($resultid);
if (! isset ($types)) {
    ... error ...
}
for ($i = 0; $i < count($types); $i++) {
    $fname = key($types);
    printf("%s :\t type =  %s\n", $fname, $types[$fname]);
    next($types);
}

```

## ifx\_fieldproperties

### Name

`ifx_fieldproperties` — List of SQL fieldproperties

### Description

```
array ifx_fieldproperties (int result_id);
```

Returns an associative array with fieldnames as key and the SQL fieldproperties as data for a query with *result\_id*. Returns FALSE on error.

Returns the Informix SQL fieldproperties of every field in the query as an associative array. Properties are encoded as: "SQLTYPE;length;precision;scale;ISNULLABLE" where SQLTYPE = the Informix type like "SQLVCHAR" etc. and ISNULLABLE = "Y" or "N".

#### Example 1. Informix SQL fieldproperties

```
$properties = ifx_fielddtypes ($resultid);
if (! isset($properties)) {
    ... error ...
}
for ($i = 0; $i < count($properties); $i++) {
    $fname = key ($properties);
    printf ("%s:\t type =  %s\n", $fname, $properties[$fname]);
    next ($properties);
}
```

## ifx\_num\_fields

### Name

`ifx_num_fields` — Returns the number of columns in the query

## Description

```
int ifx_num_fields(int result_id);
```

Returns the number of columns in query for *result\_id* or FALSE on error

After preparing or executing a query, this call gives you the number of columns in the query.

## ifx\_num\_rows

### Name

*ifx\_num\_rows* — Count the rows already fetched a query

## Description

```
int ifx_num_rows(int result_id);
```

Gives the number of rows fetched so far for a query with *result\_id* after a *ifx\_query* or *ifx\_do* query.

## ifx\_free\_result

### Name

*ifx\_free\_result* — Releases resources for the query

## Description

```
int ifx_free_result(int result_id);
```

Releases resources for the query associated with *result\_id*. Returns FALSE on error.

## ifx\_create\_char

### Name

`ifx_create_char` — Creates an char object

### Description

```
int ifx_create_char (string param);
```

Creates an char object. *param* should be the char content.

## ifx\_free\_char

### Name

`ifx_free_char` — Deletes the char object

### Description

```
int ifx_free_char (int bid);
```

Deletes the charobject for the given char object-id *bid*. Returns FALSE on error otherwise TRUE.

## ifx\_update\_char

### Name

`ifx_update_char` — Updates the content of the char object

### Description

```
int ifx_update_char(int bid, string content);
```

Updates the content of the char object for the given char object *bid*. *content* is a string with new data. Returns FALSE on error otherwise TRUE.

## ifx\_get\_char

### Name

`ifx_get_char` — Return the content of the char object

### Description

```
int ifx_get_char(int bid);
```

Returns the content of the char object for the given char object-id *bid*.

## ifx\_create\_blob

### Name

`ifx_create_blob` — Creates an blob object

### Description

```
int ifx_create_blob(int type, int mode, string param);
```

Creates an blob object.

type: 1 = TEXT, 0 = BYTE

mode: 0 = blob-object holds the content in memory, 1 = blob-object holds the content in file.

param: if mode = 0: pointer to the content, if mode = 1: pointer to the filestring.

Return FALSE on error, otherwise the new blob object-id.

## ifx\_copy\_blob

### Name

`ifx_copy_blob` — Duplicates the given blob object

### Description

```
int ifx_copy_blob(int bid);
```

Duplicates the given blob object. *bid* is the ID of the blob object.

Returns FALSE on error otherwise the new blob object-id.

## ifx\_free\_blob

### Name

`ifx_free_blob` — Deletes the blob object

### Description

```
int ifx_free_blob(int bid);
```

Deletes the blobobject for the given blob object-id *bid*. Returns FALSE on error otherwise TRUE.

## ifx\_get\_blob

### Name

`ifx_get_blob` — Return the content of a blob object

### Description

```
int ifx_get_blob(int bid);
```

Returns the content of the blob object for the given blob object-id *bid*.

## ifx\_update\_blob

### Name

`ifx_update_blob` — Updates the content of the blob object

### Description

```
ifx_update_blob(int bid, string content);
```

Updates the content of the blob object for the given blob object *bid*. *content* is a string with new data. Returns FALSE on error otherwise TRUE.

## ifx\_blobinfile\_mode

### Name

`ifx_blobinfile_mode` — Set the default blob mode for all select queries

### Description

```
void ifx_blobinfile_mode(int mode);
```

Set the default blob mode for all select queries. Mode "0" means save Byte-Blobs in memory, and mode "1" means save Byte-Blobs in a file.

## ifx\_textasvarchar

### Name

ifx\_textasvarchar — Set the default text mode

### Description

```
void ifx_textasvarchar(int mode);
```

Sets the default text mode for all select-queries. Mode "0" will return a blob id, and mode "1" will return a varchar with text content.

## ifx\_byteasvarchar

### Name

ifx\_byteasvarchar — Set the default byte mode

### Description

```
void ifx_byteasvarchar(int mode);
```

Sets the default byte mode for all select-queries. Mode "0" will return a blob id, and mode "1" will return a varchar with text content.

## ifx\_nullformat

### Name

`ifx_nullformat` — Sets the default return value on a fetch row

### Description

```
void ifx_nullformat (int mode);
```

Sets the default return value of a NULL-value on a fetch row. Mode "0" returns "", and mode "1" returns "NULL".

## ifxus\_create\_slob

### Name

`ifxus_create_slob` — Creates an slob object and opens it

### Description

```
int ifxus_create_slob (int mode);
```

Creates an slob object and opens it. Modes: 1 = LO\_RDONLY, 2 = LO\_WRONLY, 4 = LO\_APPEND, 8 = LO\_RDWR, 16 = LO\_BUFFER, 32 = LO\_NOBUFFER -> or-mask. You can also use constants named IFX\_LO\_RDONLY, IFX\_LO\_WRONLY etc. Return FALSE on error otherwise the new slob object-id.

## ifx\_free\_slob

### Name

`ifx_free_slob` — Deletes the slob object

### Description

```
int ifxus_free_slob(int bid);
```

Deletes the slob object. *bid* is the Id of the slob object. Returns FALSE on error otherwise TRUE.

## ifxus\_close\_slob

### Name

`ifxus_close_slob` — Deletes the slob object

### Description

```
int ifxus_close_slob(int bid);
```

Deletes the slob object on the given slob object-id *bid*. Return FALSE on error otherwise TRUE.

## ifxus\_open\_slob

### Name

`ifxus_open_slob` — Opens an slob object

### Description

```
int ifxus_open_slob(long bid, int mode);
```

Opens an slob object. *bid* should be an existing slob id. Modes: 1 = LO\_RDONLY, 2 = LO\_WRONLY, 4 = LO\_APPEND, 8 = LO\_RDWR, 16 = LO\_BUFFER, 32 = LO\_NOBUFFER -> or-mask. Returns FALSE on error otherwise the new slob object-id.

## ifxus\_tell\_slob

### Name

`ifxus_tell_slob` — Returns the current file or seek position

### Description

```
int ifxus_tell_slob(long bid);
```

Returns the current file or seek position of an open slob object *bid* should be an existing slob id. Return FALSE on error otherwise the seek position.

## ifxus\_seek\_slob

### Name

`ifxus_seek_slob` — Sets the current file or seek position

### Description

```
int ifxus_seek_slob(long bid, int mode, long offset);
```

Sets the current file or seek position of an open slob object. *bid* should be an existing slob id. Modes: 0 = `LO_SEEK_SET`, 1 = `LO_SEEK_CUR`, 2 = `LO_SEEK_END` and *offset* is a byte offset. Return `FALSE` on error otherwise the seek position.

## ifxus\_read\_slob

### Name

`ifxus_read_slob` — Reads *nbytes* of the slob object

### Description

```
int ifxus_read_slob(long bid, long nbytes);
```

Reads *nbytes* of the slob object. *bid* is an existing slob id and *nbytes* is the number of bytes to read. Return `FALSE` on error otherwise the string.

## ifxus\_write\_slob

### Name

`ifxus_write_slob` — Writes a string into the slob object

### Description

```
int ifxus_write_slob(long bid, string content);
```

Writes a string into the slob object. *bid* is a existing slob id and *content* the content to write. Return FALSE on error otherwise bytes written.

## **XXV. InterBase functions**

## ibase\_connect

### Name

ibase\_connect —

### Description

```
ibase_connect ( );
```

## ibase\_pconnect

### Name

ibase\_pconnect —

### Description

```
ibase_pconnect ( );
```

## ibase\_close

### Name

ibase\_close —

## Description

```
ibase_close( );
```

## ibase\_query

### Name

```
ibase_query —
```

### Description

```
ibase_query( );
```

## ibase\_fetch\_row

### Name

```
ibase_fetch_row —
```

### Description

```
ibase_fetch_row( );
```

## ibase\_free\_result

### Name

ibase\_free\_result —

### Description

```
ibase_free_result ( );
```

## ibase\_prepare

### Name

ibase\_prepare —

### Description

```
ibase_prepare ( );
```

## ibase\_bind

### Name

ibase\_bind —

## Description

```
ibase_bind( );
```

## ibase\_execute

### Name

```
ibase_execute —
```

### Description

```
ibase_execute ( );
```

## ibase\_free\_query

### Name

```
ibase_free_query —
```

### Description

```
ibase_free_query ( );
```

## **ibase\_timefmt**

### **Name**

`ibase_timefmt` —

### **Description**

```
ibase_timefmt ( );
```

# XXVI. LDAP functions

## Introduction to LDAP

LDAP is the Lightweight Directory Access Protocol, and is a protocol used to access "Directory Servers". The Directory is a special kind of database that holds information in a tree structure.

The concept is similar to your hard disk directory structure, except that in this context, the root directory is "The world" and the first level subdirectories are "countries". Lower levels of the directory structure contain entries for companies, organisations or places, while yet lower still we find directory entries for people, and perhaps equipment or documents.

To refer to a file in a subdirectory on your hard disk, you might use something like

```
/usr/local/myapp/docs
```

The forwards slash marks each division in the reference, and the sequence is read from left to right.

The equivalent to the fully qualified file reference in LDAP is the "distinguished name", referred to simply as "dn". An example dn might be.

```
cn=John Smith,ou=Accounts,o=My Company,c=US
```

The comma marks each division in the reference, and the sequence is read from right to left. You would read this dn as ..

```
country = US  
organization = My Company  
organizationalUnit = Accounts  
commonName = John Smith
```

In the same way as there are no hard rules about how you organise the directory structure of a hard disk, a directory server manager can set up any structure that is meaningful for the purpose. However, there are some conventions that are used. The message is that you can not write code to access a directory server unless you know something about its structure, any more than you can use a database without some knowledge of what is available.

## Complete code example

Retrieve information for all entries where the surname starts with "S" from a directory server, displaying an extract with name and email address.

**Example 1. LDAP search example**

```

<?php
// basic sequence with LDAP is connect, bind, search, interpret search
// result, close connection

echo "<h3>LDAP query test</h3>";
echo "Connecting ...";
$ds=ldap_connect("localhost"); // must be a valid LDAP server!
echo "connect result is ".$ds."<p>";

if ($ds) {
    echo "Binding ...";
    $r=ldap_bind($ds); // this is an "anonymous" bind, typically
                      // read-only access echo "Bind result is
    echo "Bind result is ".$r."<p>";

    echo "Searching for (sn=S*) ...";
    // Search surname entry
    $sr=ldap_search($ds,"o=My Company, c=US", "sn=S*");
    echo "Search result is ".$sr."<p>";

    echo "Number of entires returned is ".ldap_count_entries($ds,$sr)."<p>";

    echo "Getting entries ...<p>";
    $info = ldap_get_entries($ds, $sr);
    echo "Data for ".$info["count"]." items returned:<p>";

    for ($i=0; $i<$info["count"]; $i++) {
        echo "dn is: ". $info[$i]["dn"] ."<br>";
        echo "first cn entry is: ". $info[$i]["cn"][0] ."<br>";
        echo "first email entry is: ". $info[$i]["mail"][0] ."<p>";
    }

    echo "Closing connection";
    ldap_close($ds);
} else {
    echo "<h4>Unable to connect to LDAP server</h4>";
}
?>

```

## Using the PHP LDAP calls

You will need to get and compile LDAP client libraries from either the University of Michigan ldap-3.3 package or the Netscape Directory SDK. You will also need to recompile PHP with LDAP support enabled before PHP's LDAP calls will work.

Before you can use the LDAP calls you will need to know ..

- The name or address of the directory server you will use
- The "base dn" of the server (the part of the world directory that is held on this server, which could be "o=My Company,c=US")
- Whether you need a password to access the server (many servers will provide read access for an "anonymous bind" but require a password for anything else)

The typical sequence of LDAP calls you will make in an application will follow this pattern:

```
ldap_connect() // establish connection to server
|
ldap_bind()    // anonymous or authenticated "login"
|
do something like search or update the directory
and display the results
|
ldap_close()   // "logout"
```

## More Information

Lots of information about LDAP can be found at

- Netscape (<http://developer.netscape.com/tech/directory/>)
- University of Michigan (<http://www.umich.edu/~dirsvcs/ldap/index.html>)
- OpenLDAP Project (<http://www.openldap.com/>)
- LDAP World (<http://elvira.innosoft.com/ldapworld>)

The Netscape SDK contains a helpful Programmer's Guide in .html format.

# ldap\_add

## Name

ldap\_add — Add entries to LDAP directory

## Description

```
int ldap_add(int link_identifier, string dn, array entry);
```

returns true on success and false on error.

The ldap\_add function is used to add entries in the LDAP directory. The DN of the entry to be added is specified by dn. Array entry specifies the information about the entry. The values in the entries are indexed by individual attributes. In case of multiple values for an attribute, they are indexed using integers starting with 0.

```
entry["attribute1"] = value
entry["attribute2"][0] = value1
entry["attribute2"][1] = value2
```

### Example 1. Complete example with authenticated bind

```
<?php
$ds=ldap_connect("localhost"); // assuming the LDAP server is on this host

if ($ds) {
    // bind with appropriate dn to give update access
    $r=ldap_bind($ds,"cn=root, o=My Company, c=US", "secret");

    // prepare data
    $info["cn"]="John Jones";
    $info["sn"]="Jones";
    $info["mail"]="jonj@here.and.now";
    $info["objectclass"]="person";

    // add data to directory
    $r=ldap_add($ds, "cn=John Jones, o=My Company, c=US", $info);
```

```

        ldap_close($ds);
    } else {
        echo "Unable to connect to LDAP server";
    }
?>

```

## ldap\_mod\_add

### Name

`ldap_mod_add` — Add attribute values to current attributes

### Description

```
int ldap_mod_add(int link_identifier, string dn, array entry);
```

returns true on success and false on error.

This function adds attribute(s) to the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level additions are done by the `ldap_add` function.

## ldap\_mod\_del

### Name

`ldap_mod_del` — Delete attribute values from current attributes

### Description

```
int ldap_mod_del(int link_identifier, string dn, array entry);
```

returns true on success and false on error.

This function removes attribute(s) from the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level deletions are done by the `ldap_del` function.

## ldap\_mod\_replace

### Name

`ldap_mod_replace` — Replace attribute values with new ones

### Description

```
int ldap_mod_replace(int link_identifier, string dn, array entry);
```

returns true on success and false on error.

This function replaces attribute(s) from the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level modifications are done by the `ldap_modify` function.

## ldap\_bind

### Name

`ldap_bind` — Bind to LDAP directory

### Description

```
int ldap_bind(int link_identifier, string [bind_rdn], string  
[bind_password]);
```

Binds to the LDAP directory with specified RDN and password. Returns true on success and false on error.

`ldap_bind` does a bind operation on the directory. `bind_rdn` and `bind_password` are optional. If not specified, anonymous bind is attempted.

## ldap\_close

### Name

`ldap_close` — Close link to LDAP server

### Description

```
int ldap_close(int link_identifier);
```

Returns true on success, false on error.

`ldap_close` closes the link to the LDAP server that's associated with the specified *link\_identifier*.

This call is internally identical to `ldap_unbind`. The LDAP API uses the call `ldap_unbind`, so perhaps you should use this in preference to `ldap_close`.

## ldap\_connect

### Name

`ldap_connect` — Connect to an LDAP server

## Description

```
int ldap_connect(string [hostname], int [port]);
```

Returns a positive LDAP link identifier on success, or false on error.

`ldap_connect` establishes a connection to a LDAP server on a specified *hostname* and *port*. Both the arguments are optional. If no arguments are specified then the link identifier of the already opened link will be returned. If only *hostname* is specified, then the port defaults to 389.

## ldap\_count\_entries

### Name

`ldap_count_entries` — Count the number of entries in a search

### Description

```
int ldap_count_entries(int link_identifier, int result_identifier);
```

Returns number of entries in the result or false on error.

`ldap_count_entries` returns the number of entries stored in the result of previous search operations. *result\_identifier* identifies the internal ldap result.

## ldap\_delete

### Name

`ldap_delete` — Delete an entry from a directory

## Description

```
int ldap_delete(int link_identifier, string dn);
```

Returns true on success and false on error.

`ldap_delete` function delete a particular entry in LDAP directory specified by `dn`.

## ldap\_dn2ufn

### Name

`ldap_dn2ufn` — Convert DN to User Friendly Naming format

### Description

```
string ldap_dn2ufn(string dn);
```

`ldap_dn2ufn` function is used to turn a DN into a more user-friendly form, stripping off type names.

## ldap\_explode\_dn

### Name

`ldap_explode_dn` — Splits DN into its component parts

### Description

```
array ldap_explode_dn(string dn, int with_attrib);
```

`ldap_explode_dn` function is used to split the a DN returned by `ldap_get_dn` and breaks it up into its component parts. Each part is known as Relative Distinguished Name, or RDN. `ldap_explode_dn` returns an array of all those components. *with\_attr* is used to request if the RDNs are returned with only values or their attributes as well. To get RDNs with the attributes (i.e. in attribute=value format) set *with\_attr* to 0 and to get only values set it to 1.

## ldap\_first\_attribute

### Name

`ldap_first_attribute` — Return first attribute

### Description

```
string ldap_first_attribute(int link_identifier, int result_entry_identifier,
int ber_identifier);
```

Returns the first attribute in the entry on success and false on error.

Similar to reading entries, attributes are also read one by one from a particular entry.

`ldap_first_attribute` returns the first attribute in the entry pointed by the entry identifier.

Remaining attributes are retrieved by calling `ldap_next_attribute` successively.

*ber\_identifier* is the identifier to internal memory location pointer. It is passed by reference. The same *ber\_identifier* is passed to the `ldap_next_attribute()` function, which modifies that pointer.

see also `ldap_get_attributes`

## ldap\_first\_entry

### Name

`ldap_first_entry` — Return first result id

## Description

```
int ldap_first_entry(int link_identifier, int result_identifier);
```

Returns the result entry identifier for the first entry on success and false on error.

Entries in the LDAP result are read sequentially using the `ldap_first_entry` and `ldap_next_entry` functions. `ldap_first_entry` returns the entry identifier for first entry in the result. This entry identifier is then supplied to `lap_next_entry` routine to get successive entries from the result.

see also `ldap_get_entries`.

## ldap\_free\_result

### Name

`ldap_free_result` — Free result memory

### Description

```
int ldap_free_result(int result_identifier);
```

Returns true on success and false on error.

`ldap_free_result` frees up the memory allocated internally to store the result and pointed by the `result_identifier`. All result memory will be automatically freed when the script terminates.

Typically all the memory allocated for the ldap result gets freed at the end of the script. In case the script is making successive searches which return large result sets, `ldap_free_result` could be called to keep the runtime memory usage by the script low.

# ldap\_get\_attributes

## Name

`ldap_get_attributes` — Get attributes from a search result entry

## Description

```
array ldap_get_attributes (int link_identifier, int result_entry_identifier);
```

Returns a complete entry information in a multi-dimensional array on success and false on error.

`ldap_get_attributes` function is used to simplify reading the attributes and values from an entry in the search result. The return value is a multi-dimensional array of attributes and values.

Having located a specific entry in the directory, you can find out what information is held for that entry by using this call. You would use this call for an application which "browses" directory entries and/or where you do not know the structure of the directory entries. In many applications you will be searching for a specific attribute such as an email address or a surname, and won't care what other data is held.

`return_value["count"]` = number of attributes in the entry

`return_value[0]` = first attribute

`return_value[n]` = nth attribute

`return_value["attribute"]["count"]` = number of values for attribute

`return_value["attribute"][0]` = first value of the attribute

`return_value["attribute"][i]` = ith value of the attribute

### Example 1. Show the list of attributes held for a particular directory entry

```
// $ds is the link identifier for the directory

// $sr is a valid search result from a prior call to
// one of the ldap directory search calls

$entry = ldap_first_entry($ds, $sr);

$attrs = ldap_get_attributes($ds, $entry);
```

```
echo $attrs["count"]." attributes held for this entry:<p>";

for ($i=0; $i<$attrs["count"]; $i++)
    echo $attrs[$i]."<br>";

see also ldap_first_attribute and ldap_next_attribute
```

## ldap\_get\_dn

### Name

`ldap_get_dn` — Get the DN of a result entry

### Description

```
string ldap_get_dn(int link_identifier, int result_entry_identifier);
```

Returns the DN of the result entry and false on error.

`ldap_get_dn` function is used to find out the DN of an entry in the result.

## ldap\_get\_entries

### Name

`ldap_get_entries` — Get all result entries

### Description

```
array ldap_get_entries(int link_identifier, int result_identifier);
```

Returns a complete result information in a multi-dimensional array on success and false on error.

`ldap_get_entries` function is used to simplify reading multiple entries from the result and then reading the attributes and multiple values. The entire information is returned by one function call in a multi-dimensional array. The structure of the array is as follows.

The attribute index is converted to lowercase. (Attributes are case-insensitive for directory servers, but not when used as array indices)

`return_value["count"]` = number of entries in the result

`return_value[0]` : refers to the details of first entry

`return_value[i]["dn"]` = DN of the *i*th entry in the result

`return_value[i]["count"]` = number of attributes in *i*th entry

`return_value[i][j]` = *j*th attribute in the *i*th entry in the result

`return_value[i]["attribute"]["count"]` = number of values for  
attribute in *i*th entry

`return_value[i]["attribute"][j]` = *j*th value of attribute in *i*th entry

see also `ldap_first_entry` and `ldap_next_entry`

## ldap\_get\_values

### Name

`ldap_get_values` — Get all values from a result entry

### Description

```
array ldap_get_values(int link_identifier, int result_entry_identifier,
string attribute);
```

Returns an array of values for the attribute on success and false on error.

`ldap_get_values` function is used to read all the values of the attribute in the entry in the result. entry is specified by the *result\_entry\_identifier*. The number of values can be found by indexing "count" in the resultant array. Individual values are accessed by integer index in the array. The first index is 0.

This call needs a *result\_entry\_identifier*, so needs to be preceded by one of the ldap search calls and one of the calls to get an individual entry.

You application will either be hard coded to look for certain attributes (such as "surname" or "mail") or you will have to use the `ldap_get_attributes` call to work out what attributes exist for a given entry.

LDAP allows more than one entry for an attribute, so it can, for example, store a number of email addresses for one person's directory entry all labeled with the attribute "mail"

```
return_value["count"] = number of values for attribute
return_value[0] = first value of attribute
return_value[i] = ith value of attribute
```

#### **Example 1. List all values of the "mail" attribute for a directory entry**

```
// $ds is a valid link identifier for a directory server

// $sr is a valid search result from a prior call to
//      one of the ldap directory search calls

// $entry is a valid entry identifier from a prior call to
//      one of the calls that returns a directory entry

$values = ldap_get_values($ds, $entry, "mail");

echo $values["count"]." email addresses for this entry.<p>";

for ($i=0; $i < $values["count"]; $i++)
    echo $values[$i]."<br>";
```

## ldap\_get\_values\_len

### Name

`ldap_get_values_len` — Get all binary values from a result entry

### Description

```
array ldap_get_values_len(int link_identifier, int result_entry_identifier,  
string attribute);
```

Returns an array of values for the attribute on success and false on error.

`ldap_get_values_len` function is used to read all the values of the attribute in the entry in the result. entry is specified by the *result\_entry\_identifier*. The number of values can be found by indexing "count" in the resultant array. Individual values are accessed by integer index in the array. The first index is 0.

This function is used exactly like `ldap_get_values` except that it handles binary data and not string data.

## ldap\_list

### Name

`ldap_list` — Single-level search

### Description

```
int ldap_list(int link_identifier, string base_dn, string filter, array  
[attributes]);
```

Returns a search result identifier or false on error.

`ldap_list` performs the search for a specified filter on the directory with the scope `LDAP_SCOPE_ONELEVEL`.

`LDAP_SCOPE_ONELEVEL` means that the search should only return information that is at the level immediately below the base dn given in the call. (Equivalent to typing "ls" and getting a list of files and folders in the current working directory.)

This call takes an optional fourth parameter which is an array of the attributes required. See `ldap_search` notes.

### Example 1. Produce a list of all organizational units of an organization

```
// $ds is a valid link identifier for a directory server

$basedn = "o=My Company, c=US";
$justthese = array("ou");

$sr=ldap_list($ds, $basedn, "ou=*", $justthese);

$info = ldap_get_entries($ds, $sr);

for ($i=0; $i<$info["count"]; $i++)
    echo $info[$i]["ou"][0] ;
```

## ldap\_modify

### Name

`ldap_modify` — Modify an LDAP entry

### Description

```
int ldap_modify(int link_identifier, string dn, array entry);
```

Returns true on success and false on error.

`ldap_modify` function is used to modify the existing entries in the LDAP directory. The structure of the entry is same as in `ldap_add`.

## ldap\_next\_attribute

### Name

`ldap_next_attribute` — Get the next attribute in result

### Description

```
string ldap_next_attribute(int link_identifier, int result_entry_identifier,  
int ber_identifier);
```

Returns the next attribute in an entry on success and false on error.

`ldap_next_attribute` is called to retrieve the attributes in an entry. The internal state of the pointer is maintained by the *ber\_identifier*. It is passed by reference to the function. The first call to `ldap_next_attribute` is made with the *result\_entry\_identifier* returned from `ldap_first_attribute`.

see also `ldap_get_attributes`

## ldap\_next\_entry

### Name

`ldap_next_entry` — Get next result entry

## Description

```
int ldap_next_entry(int link_identifier, int result_entry_identifier);
```

Returns entry identifier for the next entry in the result whose entries are being read starting with `ldap_first_entry`. If there are no more entries in the result then it returns false.

`ldap_next_entry` function is used to retrieve the entries stored in the result. Successive calls to the `ldap_next_entry` return entries one by one till there are no more entries. The first call to `ldap_next_entry` is made after the call to `ldap_first_entry` with the `result_identifier` as returned from the `ldap_first_entry`.

see also `ldap_get_entries`

## ldap\_read

### Name

`ldap_read` — Read an entry

### Description

```
int ldap_read(int link_identifier, string base_dn, string filter, array  
[attributes]);
```

Returns a search result identifier or false on error.

`ldap_read` performs the search for a specified filter on the directory with the scope `LDAP_SCOPE_BASE`. So it is equivalent to reading an entry from the directory.

An empty filter is not allowed. If you want to retrieve absolutely all information for this entry, use a filter of `"objectClass=*"`. If you know which entry types are used on the directory server, you might use an appropriate filter such as `"objectClass=inetOrgPerson"`.

This call takes an optional fourth parameter which is an array of the attributes required. See `ldap_search` notes.

## ldap\_search

### Name

ldap\_search — Search LDAP tree

### Description

```
int ldap_search(int link_identifier, string base_dn, string filter, array
[attributes]);
```

Returns a search result identifier or false on error.

ldap\_search performs the search for a specified filter on the directory with the scope of LDAP\_SCOPE\_SUBTREE. This is equivalent to searching the entire directory. *base\_dn* specifies the base DN for the directory.

There is a optional fourth parameter, that can be added to restrict the attributes and values returned by the server to just those required. This is much more efficient than the default action (which is to return all attributes and their associated values). The use of the fourth parameter should therefore be considered good practice.

The fourth parameter is a standard PHP string array of the required attributes, eg array("mail", "sn", "cn") Note that the "dn" is always returned irrespective of which attributes types are requested.

Note too that some directory server hosts will be configured to return no more than a preset number of entries. If this occurs, the server will indicate that it has only returned a partial results set.

The search filter can be simple or advanced, using boolean operators in the format described in the LDAP documentation (see the Netscape Directory SDK (<http://developer.netscape.com/tech/directory/>) for full information on filters).

The example below retrieves the organizational unit, surname, given name and email address for all people in "My Company" where the surname or given name contains the substring \$person. This example uses a boolean filter to tell the server to look for information in more than one attribute.

#### Example 1. LDAP search

```
// $ds is a valid link identifier for a directory server

// $person is all or part of a person's name, eg "Jo"
```

```

$dn = "o=My Company, c=US";
$filter="(|(sn=$person*)(givenname=$person*))";
$justthese = array( "ou", "sn", "givenname", "mail");

$sr=ldap_search($ds, $dn, $filter, $justthese);

$info = ldap_get_entries($ds, $sr);

print $info["count"]." entries returned<p>";

```

## ldap\_unbind

### Name

`ldap_unbind` — Unbind from LDAP directory

### Description

```
int ldap_unbind(int link_identifier);
```

Returns true on success and false on error.

`ldap_unbind` function unbinds from the LDAP directory.

## ldap\_err2str

### Name

`ldap_err2str` — Convert LDAP error number into string error message

## Description

```
string ldap_err2str(int errno);
```

returns string error message.

This function returns the string error message explaining the error number `errno`. While LDAP `errno` numbers are standardized, different libraries return different or even localized textual error messages. Never check for a specific error message text, but always use an error number to check.

see also `ldap_errno` and `ldap_error`.

### Example 1. Enumerating all LDAP error messages

```
<?php
    for($i=0; $i<100; $i++) {
        printf("Error $i: %s<br>\n", ldap_str2err($i));
    }
?>
```

## ldap\_errno

### Name

`ldap_errno` — Return the LDAP error number of the last LDAP command

## Description

```
int ldap_errno(int link_id);
```

return the LDAP error number of the last LDAP command for this link.

This function returns the standardized error number returned by the last LDAP command for the given link identifier. This number can be converted into a textual error message using `ldap_err2str`.

Unless you lower your warning level in your `php3.ini` sufficiently or prefix your LDAP commands with `@` (at) characters to suppress warning output, the errors generated will also show up in your HTML output.

### Example 1. Generating and catching an error

```
<?php
// This example contains an error, which we will catch.
$ld = ldap_connect("localhost");
$bind = ldap_bind($ld);
// syntax error in filter expression (errno 87),
// must be "objectclass=*" to work.
$res = @ldap_search($ld, "o=Myorg, c=DE", "objectclass");
if (!$res) {
    printf("LDAP-Errno: %s<br>\n", ldap_errno($ld));
    printf("LDAP-Error: %s<br>\n", ldap_error($ld));
    die("Argh!<br>\n");
}
$info = ldap_get_entries($ld, $res);
printf("%d matching entries.<br>\n", $info["count"]);
?>
```

see also `ldap_err2str` and `ldap_error`.

## ldap\_error

### Name

`ldap_error` — Return the LDAP error message of the last LDAP command

### Description

```
string ldap_error(int link_id);
```

returns string error message.

This function returns the string error message explaining the error generated by the last LDAP command for the given link identifier. While LDAP errno numbers are standardized, different libraries return different or even localized textual error messages. Never check for a specific error message text, but always use an error number to check.

Unless you lower your warning level in your `php3.ini` sufficiently or prefix your LDAP commands with @ (at) characters to suppress warning output, the errors generated will also show up in your HTML output.

see also `ldap_err2str` and `ldap_errno`.

## **V. Appendixes**

# Appendix A. Migrating from PHP/FI 2.0 to PHP 3.0

## About the incompatibilities in 3.0

PHP 3.0 is rewritten from the ground up. It has a proper parser that is much more robust and consistent than 2.0's. 3.0 is also significantly faster, and uses less memory. However, some of these improvements have not been possible without compatibility changes, both in syntax and functionality.

In addition, PHP's developers have tried to clean up both PHP's syntax and semantics in version 3.0, and this has also caused some incompatibilities. In the long run, we believe that these changes are for the better.

This chapter will try to guide you through the incompatibilities you might run into when going from PHP/FI 2.0 to PHP 3.0 and help you resolve them. New features are not mentioned here unless necessary.

A conversion program that can automatically convert your old PHP/FI 2.0 scripts exists. It can be found in the `convertor` subdirectory of the PHP 3.0 distribution. This program only catches the syntax changes though, so you should read this chapter carefully anyway.

## Start/end tags

The first thing you probably will notice is that PHP's start and end tags have changed. The old `<? >` form has been replaced by three new possible forms:

### Example A-1. Migration: old start/end tags

```
<? echo "This is PHP/FI 2.0 code.\n"; >
```

As of version 2.0, PHP/FI also supports this variation:

### Example A-2. Migration: first new start/end tags

```
<? echo "This is PHP 3.0 code!\n"; ?>
```

Notice that the end tag now consists of a question mark and a greater-than character instead of just greater-than. However, if you plan on using XML on your server, you will get problems with the first

new variant, because PHP may try to execute the XML markup in XML documents as PHP code. Because of this, the following variation was introduced:

**Example A-3. Migration: second new start/end tags**

```
<?php echo "This is PHP 3.0 code!\n"; ?>
```

Some people have had problems with editors that don't understand the processing instruction tags at all. Microsoft FrontPage is one such editor, and as a workaround for these, the following variation was introduced as well:

**Example A-4. Migration: third new start/end tags**

```
<script language="php">

    echo "This is PHP 3.0 code!\n";

</script>
```

## if..endif syntax

The 'alternative' way to write if/elseif/else statements, using if(); elseif(); else; endif; cannot be efficiently implemented without adding a large amount of complexity to the 3.0 parser. Because of this, the syntax has been changed:

**Example A-5. Migration: old if..endif syntax**

```
if ($foo);
    echo "yep\n";
elseif ($bar);
    echo "almost\n";
else;
    echo "nope\n";
endif;
```

**Example A-6. Migration: new if..endif syntax**

```
if ($foo):
    echo "yep\n";
```

```
elseif ($bar):  
    echo "almost\n";  
else:  
    echo "nope\n";  
endif;
```

Notice that the semicolons have been replaced by colons in all statements but the one terminating the expression (endif).

## while syntax

Just like with if..endif, the syntax of while..endwhile has changed as well:

### Example A-7. Migration: old while..endwhile syntax

```
while ($more_to_come);  
    ...  
endwhile;
```

### Example A-8. Migration: new while..endwhile syntax

```
while ($more_to_come):  
    ...  
endwhile;
```

### Warning

If you use the old while..endwhile syntax in PHP 3.0, you will get a never-ending loop.

## Expression types

PHP/FI 2.0 used the left side of expressions to determine what type the result should be. PHP 3.0 takes both sides into account when determining result types, and this may cause 2.0 scripts to behave unexpectedly in 3.0.

Consider this example:

```
$a[0]=5;
$a[1]=7;

$key = key($a);
while ("" != $key) {
    echo "$keyn";
    next($a);
}
```

In PHP/FI 2.0, this would display both of `$a`'s indices. In PHP 3.0, it wouldn't display anything. The reason is that in PHP 2.0, because the left argument's type was string, a string comparison was made, and indeed `""` does not equal `"0"`, and the loop went through. In PHP 3.0, when a string is compared with an integer, an integer comparison is made (the string is converted to an integer). This results in comparing `atoi("")` which is 0, and `variablelist` which is also 0, and since `0==0`, the loop doesn't go through even once.

The fix for this is simple. Replace the while statement with:

```
while ((string)$key != "") {
```

## Error messages have changed

PHP 3.0's error messages are usually more accurate than 2.0's were, but you no longer get to see the code fragment causing the error. You will be supplied with a file name and a line number for the error, though.

## Short-circuited boolean evaluation

In PHP 3.0 boolean evaluation is short-circuited. This means that in an expression like `(1 || test_me())`, the function `test_me` would not be executed since nothing can change the result of the expression after the 1.

This is a minor compatibility issue, but may cause unexpected side-effects.

## Function true/false return values

Most internal functions have been rewritten so they return `TRUE` when successful and `FALSE` when failing, as opposed to 0 and -1 in PHP/FI 2.0, respectively. The new behaviour allows for more logical

code, like `$fp = fopen("/your/file") or fail("darn!");`. Because PHP/FI 2.0 had no clear rules for what functions should return when they failed, most such scripts will probably have to be checked manually after using the 2.0 to 3.0 convertor.

**Example A-9. Migration from 2.0: return values, old code**

```
$fp = fopen($file, "r");
if ($fp == -1);
    echo("Could not open $file for reading<br>\n");
endif;
```

**Example A-10. Migration from 2.0: return values, new code**

```
$fp = @fopen($file, "r") or print("Could not open $file for reading<br>\n");
```

## Other incompatibilities

- The PHP 3.0 Apache module no longer supports Apache versions prior to 1.2. Apache 1.2 or later is required.
- `echo` no longer supports a format string. Use the `printf` function instead.
- In PHP/FI 2.0, an implementation side-effect caused `$foo[0]` to have the same effect as `$foo`. This is not true for PHP 3.0.
- Reading arrays with `$array[ ]` is no longer supported

That is, you cannot traverse an array by having a loop that does `$data = $array[ ]`. Use `current` and `next` instead.

Also, `$array1[ ] = $array2` does not append the values of `$array2` to `$array1`, but appends `$array2` as the last entry of `$array1`. See also multidimensional array support.

- `"+"` is no longer overloaded as a concatenation operator for strings, instead it converts it's arguments to numbers and performs numeric addition. Use `"."` instead.

**Example A-11. Migration from 2.0: concatenation for strings**

```
echo "1" + "1";
```

In PHP 2.0 this would echo 11, in PHP 3.0 it would echo 2. Instead use:

```
echo "1"."1";  
$a = 1;  
$b = 1;  
echo $a + $b;
```

This would echo 2 in both PHP 2.0 and 3.0.

```
$a = 1;  
$b = 1;  
echo $a.$b;
```

This will echo 11 in PHP 3.0.

# Appendix B. PHP development

## Adding functions to PHP3

### Function Prototype

All functions look like this:

```
void php3_foo(INTERNAL_FUNCTION_PARAMETERS) {  
  
}
```

Even if your function doesn't take any arguments, this is how it is called.

### Function Arguments

Arguments are always of type pval. This type contains a union which has the actual type of the argument. So, if your function takes two arguments, you would do something like the following at the top of your function:

#### Example B-1. Fetching function arguments

```
pval *arg1, *arg2;  
if (ARG_COUNT(ht) != 2 || getParameters(ht,2,&arg1,&arg2)==FAILURE) {  
    WRONG_PARAM_COUNT;  
}
```

NOTE: Arguments can be passed either by value or by reference. In both cases you will need to pass `&(pval *)` to `getParameters`. If you want to check if the n'th parameter was sent to you by reference or not, you can use the function, `ParameterPassedByReference(ht,n)`. It will return either 1 or 0.

When you change any of the passed parameters, whether they are sent by reference or by value, you can either start over with the parameter by calling `pval_destructor` on it, or if it's an `ARRAY` you want to add to, you can use functions similar to the ones in `internal_functions.h` which manipulate `return_value` as an `ARRAY`.

Also if you change a parameter to `IS_STRING` make sure you first assign the new `estrdup()`'ed string and the string length, and only later change the type to `IS_STRING`. If you change the string of a parameter which already `IS_STRING` or `IS_ARRAY` you should run `pval_destructor` on it first.

## Variable Function Arguments

A function can take a variable number of arguments. If your function can take either 2 or 3 arguments, use the following:

### Example B-2. Variable function arguments

```
pval *arg1, *arg2, *arg3;
int arg_count = ARG_COUNT(ht);

if (arg_count < 2 || arg_count > 3 ||
    getParameters(ht, arg_count, &arg1, &arg2, &arg3) == FAILURE) {
    WRONG_PARAM_COUNT;
}
```

## Using the Function Arguments

The type of each argument is stored in the `pval` type field. This type can be any of the following:

**Table B-1. PHP Internal Types**

<code>IS_STRING</code>	String
<code>IS_DOUBLE</code>	Double-precision floating point
<code>IS_LONG</code>	Long integer
<code>IS_ARRAY</code>	Array
<code>IS_EMPTY</code>	None
<code>IS_USER_FUNCTION</code>	??
<code>IS_INTERNAL_FUNCTION</code>	?? (if some of these cannot be passed to a function - delete)
<code>IS_CLASS</code>	??

IS_OBJECT	??
-----------	----

If you get an argument of one type and would like to use it as another, or if you just want to force the argument to be of a certain type, you can use one of the following conversion functions:

```
convert_to_long(arg1);
convert_to_double(arg1);
convert_to_string(arg1);
convert_to_boolean_long(arg1); /* If the string is "" or "0" it be-
comes 0, 1 otherwise */
convert_string_to_number(arg1); /* Converts string to either LONG or DOU-
BLE depending on string */
```

These function all do in-place conversion. They do not return anything.

The actual argument is stored in a union; the members are:

- IS\_STRING: `arg1->value.str.val`
- IS\_LONG: `arg1->value.lval`
- IS\_DOUBLE: `arg1->value.dval`

## Memory Management in Functions

Any memory needed by a function should be allocated with either `emalloc()` or `estrdup()`. These are memory handling abstraction functions that look and smell like the normal `malloc()` and `strdup()` functions. Memory should be freed with `efree()`.

There are two kinds of memory in this program: memory which is returned to the parser in a variable, and memory which you need for temporary storage in your internal function. When you assign a string to a variable which is returned to the parser you need to make sure you first allocate the memory with either `emalloc()` or `estrdup()`. This memory should NEVER be freed by you, unless you later in the same function overwrite your original assignment (this kind of programming practice is not good though).

For any temporary/permanent memory you need in your functions/library you should use the three `emalloc()`, `estrdup()`, and `efree()` functions. They behave EXACTLY like their counterpart functions. Anything you `emalloc()` or `estrdup()` you have to `efree()` at some point or another, unless it's supposed to stick around until the end of the program; otherwise, there will be a memory leak. The meaning of "the functions behave exactly like their counterparts" is: if you `efree()` something which was not `emalloc()`'ed

nor `estrdup()`'ed you might get a segmentation fault. So please take care and free all of your wasted memory.

If you compile with `"-DDEBUG"`, PHP3 will print out a list of all memory that was allocated using `emalloc()` and `estrdup()` but never freed with `efree()` when it is done running the specified script.

## Setting Variables in the Symbol Table

A number of macros are available which make it easier to set a variable in the symbol table:

- `SET_VAR_STRING(name,value)` <sup>1</sup>
- `SET_VAR_DOUBLE(name,value)`
- `SET_VAR_LONG(name,value)`

<sup>1</sup>

Symbol tables in PHP 3.0 are implemented as hash tables. At any given time, `&symbol_table` is a pointer to the 'main' symbol table, and `active_symbol_table` points to the currently active symbol table (these may be identical like in startup, or different, if you're inside a function).

The following examples use `'active_symbol_table'`. You should replace it with `&symbol_table` if you specifically want to work with the 'main' symbol table. Also, the same functions may be applied to arrays, as explained below.

### Example B-3. Checking whether \$foo exists in a symbol table

```
if (hash_exists(active_symbol_table,"foo",sizeof("foo"))) { exists... }
else { doesn't exist }
```

### Example B-4. Finding a variable's size in a symbol table

```
hash_find(active_symbol_table,"foo",sizeof("foo",&pvalue);
check(pvalue.type);
```

Arrays in PHP 3.0 are implemented using the same hashtables as symbol tables. This means the two above functions can also be used to check variables inside arrays.

If you want to define a new array in a symbol table, you should do the following.

First, you may want to check whether it exists and abort appropriately, using `hash_exists()` or `hash_find()`.  
Next, initialize the array:

#### Example B-5. Initializing a new array

```
pval arr;

if (array_init(&arr) == FAILURE) { failed... };
hash_update(active_symbol_table, "foo", sizeof("foo"), &arr, sizeof(pval), NULL);
```

This code declares a new array, named `$foo`, in the active symbol table. This array is empty.

Here's how to add new entries to it:

#### Example B-6. Adding entries to a new array

```
pval entry;

entry.type = IS_LONG;
entry.value.lval = 5;

/* defines $foo["bar"] = 5 */
hash_update(arr.value.ht, "bar", sizeof("bar"), &entry, sizeof(pval), NULL);

/* defines $foo[7] = 5 */
hash_index_update(arr.value.ht, 7, &entry, sizeof(pval), NULL);

/* defines the next free place in $foo[],
 * $foo[8], to be 5 (works like php2)
 */
hash_next_index_insert(arr.value.ht, &entry, sizeof(pval), NULL);
```

If you'd like to modify a value that you inserted to a hash, you must first retrieve it from the hash. To prevent that overhead, you can supply a `pval **` to the hash add function, and it'll be updated with the `pval *` address of the inserted element inside the hash. If that value is `NULL` (like in all of the above examples) - that parameter is ignored.

`hash_next_index_insert()` uses more or less the same logic as `"$foo[] = bar;"` in PHP 2.0.

If you are building an array to return from a function, you can initialize the array just like above by doing:

```
if (array_init(return_value) == FAILURE) { failed...; }
```

...and then adding values with the helper functions:

```
add_next_index_long(return_value,long_value);
add_next_index_double(return_value,double_value);
add_next_index_string(return_value,estrdup(string_value));
```

Of course, if the adding isn't done right after the array initialization, you'd probably have to look for the array first:

```
pval *arr;

if (hash_find(active_symbol_table,"foo",sizeof("foo"),(void **)&arr)==FAILURE) { can't find
else { use arr->value.ht... }
```

Note that `hash_find` receives a pointer to a pval pointer, and not a pval pointer.

Just about any hash function returns SUCCESS or FAILURE (except for `hash_exists()`, which returns a boolean truth value).

## Returning simple values

A number of macros are available to make returning values from a function easier.

The `RETURN_*` macros all set the return value and return from the function:

- `RETURN`
- `RETURN_FALSE`
- `RETURN_TRUE`
- `RETURN_LONG(l)`
- `RETURN_STRING(s,dup)` If `dup` is true, duplicates the string
- `RETURN_STRINGL(s,l,dup)` Return string (s) specifying length (l).
- `RETURN_DOUBLE(d)`

The `RETVAL_*` macros set the return value, but do not return.

- `RETVAL_FALSE`

- RETVAL\_TRUE
- RETVAL\_LONG(l)
- RETVAL\_STRING(s,dup) If dup is true, duplicates the string
- RETVAL\_STRINGL(s,l,dup) Return string (s) specifying length (l).
- RETVAL\_DOUBLE(d)

The string macros above will all `estrdup()` the passed 's' argument, so you can safely free the argument after calling the macro, or alternatively use statically allocated memory.

If your function returns boolean success/error responses, always use `RETURN_TRUE` and `RETURN_FALSE` respectively.

## Returning complex values

Your function can also return a complex data type such as an object or an array.

Returning an object:

1. Call `object_init(return_value)`.
2. Fill it up with values. The functions available for this purpose are listed below.
3. Possibly, register functions for this object. In order to obtain values from the object, the function would have to fetch "this" from the `active_symbol_table`. Its type should be `IS_OBJECT`, and it's basically a regular hash table (i.e., you can use regular hash functions on `.value.ht`). The actual registration of the function can be done using:

```
add_method( return_value, function_name, function_ptr );
```

The functions used to populate an object are:

- `add_property_long( return_value, property_name, l )` - Add a property named 'property\_name', of type long, equal to 'l'
- `add_property_double( return_value, property_name, d )` - Same, only adds a double
- `add_property_string( return_value, property_name, str )` - Same, only adds a string
- `add_property_stringl( return_value, property_name, str, l )` - Same, only adds a string of length 'l'

Returning an array:

1. Call `array_init(return_value)`.
2. Fill it up with values. The functions available for this purpose are listed below.

The functions used to populate an array are:

- `add_assoc_long(return_value, key, l)` - add associative entry with key 'key' and long value 'l'
- `add_assoc_double(return_value, key, d)`
- `add_assoc_string(return_value, key, str, duplicate)`
- `add_assoc_stringl(return_value, key, str, length, duplicate)` specify the string length
- `add_index_long(return_value, index, l)` - add entry in index 'index' with long value 'l'
- `add_index_double(return_value, index, d)`
- `add_index_string(return_value, index, str)`
- `add_index_stringl(return_value, index, str, length)` - specify the string length
- `add_next_index_long(return_value, l)` - add an array entry in the next free offset with long value 'l'
- `add_next_index_double(return_value, d)`
- `add_next_index_string(return_value, str)`
- `add_next_index_stringl(return_value, str, length)` - specify the string length

## Using the resource list

PHP 3.0 has a standard way of dealing with various types of resources. This replaces all of the local linked lists in PHP 2.0.

Available functions:

- `php3_list_insert(ptr, type)` - returns the 'id' of the newly inserted resource
- `php3_list_delete(id)` - delete the resource with the specified id
- `php3_list_find(id, *type)` - returns the pointer of the resource with the specified id, updates 'type' to the resource's type

Typically, these functions are used for SQL drivers but they can be used for anything else; for instance, maintaining file descriptors.

Typical list code would look like this:

**Example B-7. Adding a new resource**

```
RESOURCE *resource;

/* ...allocate memory for resource and acquire resource... */
/* add a new resource to the list */
return_value-
>value.lval = php3_list_insert((void *) resource, LE_RESOURCE_TYPE);
return_value->type = IS_LONG;
```

**Example B-8. Using an existing resource**

```
pval *resource_id;
RESOURCE *resource;
int type;

convert_to_long(resource_id);
resource = php3_list_find(resource_id->value.lval, &type);
if (type != LE_RESOURCE_TYPE) {
    php3_error(E_WARNING, "resource index %d has the wrong type", resource_id-
>value.lval);
    RETURN_FALSE;
}
/* ...use resource... */
```

**Example B-9. Deleting an existing resource**

```
pval *resource_id;
RESOURCE *resource;
int type;

convert_to_long(resource_id);
php3_list_delete(resource_id->value.lval);
```

The resource types should be registered in `php3_list.h`, in enum `list_entry_type`. In addition, one should add shutdown code for any new resource type defined, in `list.c`'s `list_entry_destructor()` (even if you don't have anything to do on shutdown, you must add an empty case).

## Using the persistent resource table

PHP 3.0 has a standard way of storing persistent resources (i.e., resources that are kept in between hits). The first module to use this feature was the MySQL module, and mSQL followed it, so one can get the general impression of how a persistent resource should be used by reading `mysql.c`. The functions you should look at are:

```
php3_mysql_do_connect
php3_mysql_connect()
php3_mysql_pconnect()
```

The general idea of persistence modules is this:

1. Code all of your module to work with the regular resource list mentioned in section (9).
2. Code extra connect functions that check if the resource already exists in the persistent resource list. If it does, register it as in the regular resource list as a pointer to the persistent resource list (because of 1., the rest of the code should work immediately). If it doesn't, then create it, add it to the persistent resource list AND add a pointer to it from the regular resource list, so all of the code would work since it's in the regular resource list, but on the next connect, the resource would be found in the persistent resource list and be used without having to recreate it. You should register these resources with a different type (e.g. `LE_MYSQL_LINK` for non-persistent link and `LE_MYSQL_PLINK` for a persistent link).

If you read `mysql.c`, you'll notice that except for the more complex connect function, nothing in the rest of the module has to be changed.

The very same interface exists for the regular resource list and the persistent resource list, only 'list' is replaced with 'plist':

- `php3_plist_insert(ptr, type)` - returns the 'id' of the newly inserted resource
- `php3_plist_delete(id)` - delete the resource with the specified id
- `php3_plist_find(id,*type)` - returns the pointer of the resource with the specified id, updates 'type' to the resource's type

However, it's more than likely that these functions would prove to be useless for you when trying to implement a persistent module. Typically, one would want to use the fact that the persistent resource list is really a hash table. For instance, in the MySQL/mSQL modules, when there's a `pconnect()` call (persistent connect), the function builds a string out of the host/user/passwd that were passed to the function, and hashes the SQL link with this string as a key. The next time someone calls a `pconnect()` with the same host/user/passwd, the same key would be generated, and the function would find the SQL link in the persistent list.

Until further documented, you should look at `mysql.c` or `msql.c` to see how one should use the plist's hash table abilities.

One important thing to note: resources going into the persistent resource list must *\*NOT\** be allocated with PHP's memory manager, i.e., they should NOT be created with `emalloc()`, `estrdup()`, etc. Rather, one should use the regular `malloc()`, `strdup()`, etc. The reason for this is simple - at the end of the request (end of the hit), every memory chunk that was allocated using PHP's memory manager is deleted. Since the persistent list isn't supposed to be erased at the end of a request, one mustn't use PHP's memory manager for allocating resources that go to it.

When you register a resource that's going to be in the persistent list, you should add destructors to it both in the non-persistent list and in the persistent list. The destructor in the non-persistent list shouldn't do anything. The one in the persistent list destructor should properly free any resources obtained by that type (e.g. memory, SQL links, etc). Just like with the non-persistent resources, you *\*MUST\** add destructors for every resource, even it requires no destructotion and the destructor would be empty. Remember, since `emalloc()` and friends aren't to be used in conjunction with the persistent list, you mustn't use `efree()` here either.

## Adding runtime configuration directives

Many of the features of PHP3 can be configured at runtime. These configuration directives can appear in either the designated `php3.ini` file, or in the case of the Apache module version in the Apache `.conf` files. The advantage of having them in the Apache `.conf` files is that they can be configured on a per-directory basis. This means that one directory may have a certain `safemodeexecdir` for example, while another directory may have another. This configuration granularity is especially handy when a server supports multiple virtual hosts.

The steps required to add a new directive:

1. Add directive to `php3_ini_structure` struct in `mod_php3.h`.
2. In `main.c`, edit the `php3_module_startup` function and add the appropriate `cfg_get_string()` or `cfg_get_long()` call.
3. Add the directive, restrictions and a comment to the `php3_commands` structure in `mod_php3.c`. Note the restrictions part. `RSRC_CONF` are directives that can only be present in the actual Apache `.conf` files. Any `OR_OPTIONS` directives can be present anywhere, include normal `.htaccess` files.
4. In either `php3take1handler()` or `php3flaghandler()` add the appropriate entry for your directive.
5. In the configuration section of the `_php3_info()` function in `functions/info.c` you need to add your new directive.

6. And last, you of course have to use your new directive somewhere. It will be addressable as `php3_ini.directive`.

## Calling User Functions

To call user functions from an internal function, you should use the `call_user_function` function.

`call_user_function` returns `SUCCESS` on success, and `FAILURE` in case the function cannot be found. You should check that return value! If it returns `SUCCESS`, you are responsible for destroying the returned pval yourself (or return it as the return value of your function). If it returns `FAILURE`, the value of `retval` is undefined, and you mustn't touch it.

All internal functions that call user functions *must* be reentrant. Among other things, this means they must not use globals or static variables.

`call_user_function` takes six arguments:

### HashTable \*function\_table

This is the hash table in which the function is to be looked up.

### pval \*object

This is a pointer to an object on which the function is invoked. This should be `NULL` if a global function is called. If it's not `NULL` (i.e. it points to an object), the `function_table` argument is ignored, and instead taken from the object's hash. The object *may* be modified by the function that is invoked on it (that function will have access to it via `$this`). If for some reason you don't want that to happen, send a copy of the object instead.

### pval \*function\_name

The name of the function to call. Must be a pval of type `IS_STRING` with `function_name.str.val` and `function_name.str.len` set to the appropriate values. The `function_name` is modified by `call_user_function()` - it's converted to lowercase. If you need to preserve the case, send a copy of the function name instead.

## **pval \*retval**

A pointer to a pval structure, into which the return value of the invoked function is saved. The structure must be previously allocated - `call_user_function` does NOT allocate it by itself.

## **int param\_count**

The number of parameters being passed to the function.

## **pval \*params[]**

An array of pointers to values that will be passed as arguments to the function, the first argument being in offset 0, the second in offset 1, etc. The array is an array of pointers to pval's; The pointers are sent as-is to the function, which means if the function modifies its arguments, the original values are changed (passing by reference). If you don't want that behavior, pass a copy instead.

# **Reporting Errors**

To report errors from an internal function, you should call the `php3_error` function. This takes at least two parameters – the first is the level of the error, the second is the format string for the error message (as in a standard `printf` call), and any following arguments are the parameters for the format string. The error levels are:

## **E\_NOTICE**

Notices are not printed by default, and indicate that the script encountered something that could indicate an error, but could also happen in the normal course of running a script. For example, trying to access the value of a variable which has not been set, or calling `stat` on a file that doesn't exist.

## **E\_WARNING**

Warnings are printed by default, but do not interrupt script execution. These indicate a problem that should have been trapped by the script before the call was made. For example, calling `ereg` with an invalid regular expression.

## **E\_ERROR**

Errors are also printed by default, and execution of the script is halted after the function returns. These indicate errors that can not be recovered from, such as a memory allocation problem.

## **E\_PARSE**

Parse errors should only be generated by the parser. The code is listed here only for the sake of completeness.

## **E\_CORE\_ERROR**

This is like an E\_ERROR, except it is generated by the core of PHP. Functions should not generate this type of error.

## **E\_CORE\_WARNING**

This is like an E\_WARNING, except it is generated by the core of PHP. Functions should not generate this type of error.

## **Notes**

Be careful here. The value part must be malloc'ed manually because the memory management code will try to free this pointer later. Do not pass statically allocated memory into a SET\_VAR\_STRING.

# Appendix C. The PHP Debugger

## Using the Debugger

PHP's internal debugger is useful for tracking down evasive bugs. The debugger works by connecting to a TCP port for every time PHP starts up. All error messages from that request will be sent to this TCP connection. This information is intended for "debugging server" that can run inside an IDE or programmable editor (such as Emacs).

How to set up the debugger:

1. Set up a TCP port for the debugger in the configuration file (debugger.port) and enable it (debugger.enabled).
2. Set up a TCP listener on that port somewhere (for example **socket -l -s 1400** on UNIX).
3. In your code, run "debugger\_on(*host*)", where *host* is the IP number or name of the host running the TCP listener.

Now, all warnings, notices etc. will show up on that listener socket, *even if you them turned off with error\_reporting.*

## Debugger Protocol

The debugger protocol is line-based. Each line has a *type*, and several lines compose a *message*. Each message starts with a line of the type *start* and terminates with a line of the type *end*. PHP may send lines for different messages simultaneously.

A line has this format:

*date time host(pid) type:message-data*

*date*

Date in ISO 8601 format (*yyyy-mm-dd*)

*time*

Time including microseconds: *hh:mm:uuuuuu*

*host*

DNS name or IP address of the host where the script error was generated.

*pid*

PID (process id) on *host* of the process with the PHP script that generated this error.

*type*

Type of line. Tells the receiving program about what it should treat the following data as:

**Table C-1. Debugger Line Types**

Name	Meaning
start	Tells the receiving program that a debugger message starts here. The contents of <i>data</i> will be the type of error message, listed below.
message	The PHP error message.
location	File name and line number where the error occurred. The first location line will always contain the top-level location. <i>data</i> will contain <i>file:line</i> . There will always be a location line after message and after every function.
frames	Number of frames in the following stack dump. If there are four frames, expect information about four levels of called functions. If no "frames" line is given, the depth should be assumed to be 0 (the error occurred at top-level).
function	Name of function where the error occurred. Will be repeated once for every level in the function call stack.
end	Tells the receiving program that a debugger message ends here.

*data*

Line data.

**Table C-2. Debugger Error Types**

<b>Debugger</b>	<b>PHP Internal</b>
warning	E_WARNING
error	E_ERROR
parse	E_PARSE
notice	E_NOTICE
core-error	E_CORE_ERROR
core-warning	E_CORE_WARNING
unknown	(any other)

**Example C-1. Example Debugger Message**

```

1998-04-05 23:27:400966 lucifer.guardian.no(20481) start: notice
1998-04-05 23:27:400966 lucifer.guardian.no(20481) message: Uninitialized variable
1998-04-05 23:27:400966 lucifer.guardian.no(20481) location: (null):7
1998-04-05 23:27:400966 lucifer.guardian.no(20481) frames: 1
1998-04-05 23:27:400966 lucifer.guardian.no(20481) function: display
1998-04-05 23:27:400966 lucifer.guardian.no(20481) location: /home/ssb/public_html/test.php3:10
1998-04-05 23:27:400966 lucifer.guardian.no(20481) end: notice

```

