

PHP Manual

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Zeev Suraski

Andrei Zmievski

Jouni Ahto

Edited by

Stig Sæther Bakken

Egon Schmid

PHP Manual

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

by

Edited by Stig Sæther Bakken

Edited by Egon Schmid

Published 04-02-2001

Copyright © 1997, 1998, 1999, 2000, 2001 by the PHP Documentation Group

Copyright

This manual is © Copyright 1997, 1998, 1999, 2000, 2001 by 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.

Table of Contents

Preface	41
About this Manual.....	41
I. Getting Started	43
1. Introduction.....	43
What is PHP?	45
What can PHP do?.....	45
A brief history of PHP.....	45
2. Installation.....	47
Downloading the latest version.....	49
Installation on UNIX systems	49
Apache Module Quick Reference.....	49
Building	50
Linux installs	50
Using Packages.....	50
HP-UX installs	50
Solaris installs	51
Required software.....	51
Using Packages.....	52
Mac OS X installs	52
Using Packages.....	52
Compiling for OS X server.....	52
Complete list of configure options	53
Database.....	54
Ecommerce	58
Graphics	58
Miscellaneous	59
Networking	65
PHP Behaviour	66
Server	67
Text and language	68
XML	68
Installation on Windows 9x/Me/NT/2000 systems	69
Windows InstallShield.....	69
General Installation Steps	70
Building from source	71
Preparations.....	71
Putting it all together	71
Compiling.....	72
Installation of Windows extensions	72
Apache Server installs	73
Details of installing PHP with Apache on Unix.....	73
Details of installing PHP on Windows with Apache 1.3.x	75
CGI/Commandline Server Installs	75
Testing.....	76
Benchmarking.....	76
fhttpd Server Installs	76
Caudium Server installs.....	76
IIS/PWS Server installs	77
Windows and PWS/IIS 3	77
Windows and PWS 4 or newer	78
Windows NT/2000 and IIS 4 or newer	78
Netscape and iPlanet Enterprise Server Installs	79
OmniHTTPd Server installs	80

OmniHTTPd 2.0b1 and up for Windows.....	80
Oreilly Website Pro Server installs.....	81
Oreilly Website Pro 2.5 and up for Windows	81
Xitami Server installs	81
Xitami for Windows	81
Other web servers.....	82
Problems?.....	82
Read the FAQ.....	82
Bug reports.....	82
Other problems	82
3. Configuration	83
The configuration file	85
General Configuration Directives	85
Mail Configuration Directives	88
Safe Mode Configuration Directives	89
Debugger Configuration Directives	89
Extension Loading Directives.....	89
MySQL Configuration Directives.....	89
mSQL Configuration Directives	90
Postgres Configuration Directives	90
SESAM Configuration Directives	90
Sybase Configuration Directives	91
Sybase-CT Configuration Directives	91
Informix Configuration Directives.....	92
BC Math Configuration Directives	92
Browser Capability Configuration Directives.....	92
Unified ODBC Configuration Directives.....	92
4. Security	95
Installed as CGI binary.....	97
Possible attacks.....	97
Case 1: only public files served	97
Case 2: using –enable-force-cgi-redirect	98
Case 3: setting doc_root or user_dir	98
Case 4: PHP parser outside of web tree.....	98
Installed as an Apache module.....	99
Filesystem Security	99
Error Reporting	100
User Submitted Data	101
General considerations	102
Keeping Current	102
II. Language Reference	103
5. Basic syntax	103
Escaping from HTML	105
Instruction separation	105
Comments	105
6. Types	107
Integers	109
Floating point numbers.....	109
Strings.....	109
String conversion	111
Arrays	112
Single Dimension Arrays.....	112
Multi-Dimensional Arrays.....	112
Objects	114
Object Initialization	114

Type Juggling	114
Type Casting	115
7. Variables.....	117
Basics	119
Predefined variables	119
Apache variables.....	120
Environment variables	121
PHP variables.....	121
Variable scope	122
Variable variables	124
Variables from outside PHP	125
HTML Forms (GET and POST).....	125
IMAGE SUBMIT variable names.....	125
HTTP Cookies	125
Environment variables	126
Dots in incoming variable names.....	126
Determining variable types.....	126
8. Constants.....	127
9. Expressions	131
10. Operators.....	135
Arithmetic Operators.....	137
Assignment Operators	137
Bitwise Operators.....	137
Comparison Operators.....	138
Error Control Operators	138
Execution Operators	139
Incrementing/Decrementing Operators	139
Logical Operators.....	140
Operator Precedence	140
String Operators	141
11. Control Structures	143
if.....	145
else	145
elseif	145
Alternative syntax for control structures	146
while	146
do..while	147
for	148
foreach.....	149
break	151
continue.....	151
switch	152
require()	153
include()	154
require_once()	157
include_once()	159
12. Functions.....	161
User-defined functions	163
Function arguments	163
Making arguments be passed by reference	163
Default argument values	164
Variable-length argument lists	165
Returning values.....	165
old_function	165
Variable functions.....	165
13. Classes and Objects.....	167

<code>class</code>	169
14. References Explained.	171
What References Are	173
What References Do	173
What References Are Not	173
Passing by Reference	174
Returning References	174
Unsetting References	175
Spotting References	175
<code>global</code> References	175
<code>this</code>	175
III. Features	177
15. Error Handling	177
16. Creating and manipulating images	183
17. HTTP authentication with PHP	187
18. Cookies	191
19. Handling file uploads	195
POST method uploads	197
Common Pitfalls	198
Uploading multiple files	199
PUT method support	199
20. Using remote files	201
21. Connection handling	205
22. Persistent Database Connections	209
IV. Function Reference	213
I. Apache-specific Functions	213
<code>apache_lookup_uri</code>	215
<code>apache_note</code>	215
<code>getallheaders</code>	215
<code>virtual</code>	216
<code>ascii2ebcdic</code>	216
<code>ebcdic2ascii</code>	217
II. Array Functions	219
<code>array</code>	221
<code>array_count_values</code>	222
<code>array_diff</code>	222
<code>array_flip</code>	222
<code>array_intersect</code>	223
<code>array_keys</code>	223
<code>array_merge</code>	224
<code>array_merge_recursive</code>	224
<code>array_multisort</code>	225
<code>array_pad</code>	226
<code>array_pop</code>	226
<code>array_push</code>	227
<code>array_rand</code>	227
<code>array_reverse</code>	228
<code>array_shift</code>	228
<code>array_slice</code>	229
<code>array_splice</code>	229
<code>array_unique</code>	230
<code>array_unshift</code>	230
<code>array_values</code>	231
<code>array_walk</code>	231
<code>arsort</code>	232

asort	233
compact	233
count	234
current	234
each	235
end	236
extract	236
in_array	237
key	238
krsort	238
ksort	239
list	239
natsort	240
natcasesort	241
next	241
pos	242
prev	242
range	242
reset	242
rsort	243
shuffle	243
sizeof	244
sort	244
uasort	245
uksort	245
usort	246
III. Aspell functions	249
aspell_new	251
aspell_check	251
aspell_check_raw	251
aspell_suggest	252
IV. BCMath Arbitrary Precision Mathematics Functions	253
bcadd	255
bccomp	255
bcdiv	255
bcmod	255
bcmul	255
bcpow	256
bcscale	256
bcsqrt	256
bcsub	256
V. Calendar functions	257
JDToGregorian	259
GregorianToJD	259
JDToJulian	259
JulianToJD	259
JDToJewish	260
JewishToJD	260
JDToFrench	260
FrenchToJD	260
JDMonthName	260
JDDayOfWeek	261
easter_date	261
easter_days	262
unixtojd	262
jdtounix	263

VI. CCVS API Functions.....	265
	267
VII. COM support functions for Windows	269
com_load	271
com_invoke	271
com_propget.....	271
com_get.....	271
com_propput.....	271
com_propset.....	271
com_set.....	272
VIII. Class/Object Functions	273
call_user_method	277
class_exists	277
get_class	277
get_class_methods.....	278
get_class_vars.....	278
get_declared_classes	278
get_object_vars.....	278
get_parent_class	279
is_subclass_of.....	279
method_exists.....	280
IX. ClibPDF functions	281
cpdf_global_set_document_limits	285
cpdf_set_creator	285
cpdf_set_title	285
cpdf_set_subject	285
cpdf_set_keywords	285
cpdf_open	286
cpdf_close	286
cpdf_page_init	286
cpdf_finalize_page	286
cpdf_finalize	287
cpdf_output_buffer	287
cpdf_save_to_file	287
cpdf_set_current_page	287
cpdf_begin_text	288
cpdf_end_text	288
cpdf_show	288
cpdf_show_xy	289
cpdf_text	289
cpdf_set_font	289
cpdf_set_leading	290
cpdf_set_text_rendering	290
cpdf_set_horiz_scaling	290
cpdf_set_text_rise	290
cpdf_set_text_matrix	290
cpdf_set_text_pos	291
cpdf_set_char_spacing	291
cpdf_set_word_spacing	291
cpdf_continue_text	291
cpdf_stringwidth	291
cpdf_save	292
cpdf_restore	292
cpdf_translate	292
cpdf_scale	293
cpdf_rotate.....	293

cpdf_setflat	293
cpdf_setlinejoin	293
cpdf_setlinecap	293
cpdf_setmiterlimit	293
cpdf_setlinewidth	294
cpdf_setdash	294
cpdf_newpath	294
cpdf_moveto	294
cpdf_rmoveto	294
cpdf_curveto	295
cpdf_lineto	295
cpdf_rlineto	295
cpdf_circle	295
cpdf_arc	296
cpdf_rect	296
cpdf_closepath	296
cpdf_stroke	296
cpdf_closepath_stroke	297
cpdf_fill	297
cpdf_fill_stroke	297
cpdf_closepath_fill_stroke	297
cpdf_clip	298
cpdf_setgray_fill	298
cpdf_setgray_stroke	298
cpdf_setgray	298
cpdf_setrgbcolor_fill	298
cpdf_setrgbcolor_stroke	299
cpdf_setrgbcolor	299
cpdf_add_outline	299
cpdf_set_page_animation	300
cpdf_import_jpeg	300
cpdf_place_inline_image	300
cpdf_add_annotation	301
X. CURL, Client URL Library Functions	303
curl_init	305
curl_setopt	305
curl_exec	307
curl_close	307
curl_version	307
XI. Cybercash payment functions	309
cybercash_encr	311
cybercash_decr	311
cybercash_base64_encode	311
cybercash_base64_decode	311
XII. Database (dbm-style) abstraction layer functions	313
dba_close	315
dba_delete	315
dba_exists	315
dba_fetch	315
dba_firstkey	316
dba_insert	316
dba_nextkey	316
dba_popen	316
dba_open	317
dba_optimize	317
dba_replace	317

dba_sync	318
XIII. Date and Time functions	319
checkdate	321
date	321
getdate	322
gettimeofday	323
gdate	323
gmmktime	323
gmstrftime	323
localtime	324
microtime	324
mktime	325
strftime	326
time	327
strtotime	327
XIV. dBase functions	329
dbase_create	331
dbase_open	331
dbase_close	332
dbase_pack	332
dbase_add_record	332
dbase_replace_record	332
dbase_delete_record	332
dbase_get_record	333
dbase_get_record_with_names	333
dbase_numfields	333
dbase_numrecords	333
XV. DBM Functions	335
dbmopen	337
dbmclose	337
dbmexists	337
dbmfetch	337
dbminsert	337
dbmreplace	338
dbmdelete	338
dbmfirstkey	338
dbmnextkey	338
dblist	339
XVI. Directory functions	341
chdir	343
dir	343
closedir	343
getcwd	343
opendir	343
readdir	344
rewinddir	344
XVII. DOM XML functions	347
xml/doc	349
xml/docfile	349
xml/tree	349
dom/xml/_root	349
dom/xml/_add_root	349
dom/xml/_dumpmem	350
dom/xml/_attributes	350
dom/xml/_get_attribute	350
dom/xml/_set_attribute	350

domxml_children	350
domxml_new_child	350
domxml_new_xmldoc	351
domxml_node	351
XVIII. Error Handling and Logging Functions	353
error_log	355
error_reporting	355
restore_error_handler	357
set_error_handler	357
trigger_error	359
user_error	360
XIX. filePro functions	361
filepro	363
filepro_fieldname	363
filepro_fieldtype	363
filepro_fieldwidth	363
filepro_retrieve	363
filepro_fieldcount	363
filepro.RowCount	364
XX. Filesystem functions	365
basename	367
chgrp	367
chmod	367
chown	368
clearstatcache	368
copy	368
delete	369
dirname	369
diskfreespace	369
fclose	369
feof	370
fflush	370
fgetc	370
fgetcsv	370
fgets	371
fgetss	371
file	372
file_exists	372
fileatime	372
filectime	373
filegroup	373
fileinode	373
filemtime	374
fileowner	374
fileperms	374
filesize	374
filetype	375
flock	375
fopen	375
fpassthru	376
fputs	377
fread	377
fscanf	377
fseek	378
fstat	378
ftell	379

ftruncate	379
fwrite	379
set_file_buffer	380
is_dir	380
is_executable	380
is_file	381
is_link	381
is_readable	381
is_writable	381
is_uploaded_file	382
link	382
linkinfo	382
mkdir	383
move_uploaded_file	383
pclose	383
popen	384
readfile	384
readlink	384
rename	385
rewind	385
rmdir	385
stat	385
lstat	386
realpath	387
symlink	387
tempnam	387
tmpfile	388
touch	388
umask	388
unlink	389
XXI. Forms Data Format functions	391
fdf_open	393
fdf_close	393
fdf_create	393
fdf_save	394
fdf_get_value	394
fdf_set_value	394
fdf_next_field_name	394
fdf_set_ap	395
fdf_set_status	395
fdf_get_status	395
fdf_set_file	395
fdf_get_file	395
fdf_set_flags	396
fdf_set_opt	396
fdf_set_submit_form_action	396
fdf_set_javascript_action	396
XXII. FTP functions	399
ftp_connect	401
ftp_login	401
ftp_pwd	401
ftp_cdup	401
ftp_chdir	401
ftp_mkdir	401
ftp_rmdir	402
ftp_nlist	402

ftp_rawlist.....	402
ftp_systype.....	402
ftp_pasv.....	402
ftp_get.....	403
ftp_fget.....	403
ftp_put.....	403
ftp_fput.....	403
ftp_size.....	404
ftp_mdtm.....	404
ftp_rename.....	404
ftp_delete.....	404
ftp_site.....	405
ftp_quit.....	405
XXIII. Function Handling functions.....	407
call_user_func	409
create_function.....	409
func_get_arg.....	411
func_get_args.....	411
func_num_args.....	412
function_exists.....	412
register_shutdown_function	413
XXIV. GNU Gettext.....	415
bindtextdomain.....	417
dcgettext.....	417
dgettext.....	417
gettext.....	417
textdomain.....	417
XXV. GMP functions.....	419
gmp_init.....	421
gmp_intval.....	421
gmp_strval.....	421
gmp_add.....	422
gmp_sub.....	422
gmp_mul.....	422
gmp_div_q.....	422
gmp_div_r.....	422
gmp_div_qr.....	423
gmp_div.....	423
gmp_mod.....	423
gmp_divexact.....	424
gmp_cmp.....	424
gmp_neg.....	424
gmp_abs.....	424
gmp_sign.....	424
gmp_fact.....	424
gmp_sqrt.....	425
gmp_sqrtrem.....	425
gmp_perfect_square	425
gmp_pow.....	425
gmp_powm.....	425
gmp_prob_prime.....	426
gmp_gcd.....	426
gmp_gcdext.....	426
gmp_invert.....	426
gmp_legendre.....	426
gmp_jacobi.....	427

gmp_random.....	427
gmp_and.....	427
gmp_or	427
gmp_xor	427
gmp_setbit	428
gmp_clrbit	428
gmp_scan0.....	428
gmp_scan1.....	428
gmp_popcount.....	428
gmp_hamdist	429
XXVI. HTTP functions.....	431
header	433
headers_sent	433
setcookie	433
XXVII. Hyperwave functions.....	437
hw_Array2Objrec.....	441
hw_Children.....	441
hw_ChildrenObj	441
hw_Close.....	441
hw_Connect.....	441
hw_Cp	442
hw_Deleteobject	442
hw_DocByAnchor	442
hw_DocByAnchorObj	442
hw_Document_Attributes	442
hw_Document_BodyTag	443
hw_Document_Content	443
hw_Document_SetContent	443
hw_Document_Size	443
hw_ErrorMsg	444
hw_EditText	444
hw_Error	444
hw_Free_Document	444
hw_GetParents	445
hw_GetParentsObj	445
hw_GetChildColl	445
hw_GetChildCollObj	445
hw_GetRemote	445
hw_GetRemoteChildren	446
hw_GetSrcByDestObj	446
hw_GetObject	446
hw_GetAndLock	447
hw_GetText	447
hw_GetObjectByQuery	448
hw_GetObjectByQueryObj	448
hw_GetObjectByQueryColl	448
hw_GetObjectByQueryCollObj	448
hw_GetChildDocColl	449
hw_GetChildDocCollObj	449
hw_GetAnchors	449
hw_GetAnchorsObj	449
hw_Mv	449
hw_Identify	450
hw_InCollections	450
hw_Info	450
hw_InsColl	450

hw_InsDoc	451
hw_InsertDocument	451
hw_InsertObject	451
hw_mapid	451
hw_Modifyobject	452
hw_New_Document	454
hw_Objrec2Array	454
hw_Output_Document	454
hw_pConnect	454
hw_PipeDocument	455
hw_Root	455
hw_Unlock	455
hw_Who	455
hw_getusername	456
XXVIII. ICAP Functions	457
icap_open	459
icap_close	459
icap_fetch_event	459
icap_list_events	460
icap_store_event	460
icap_delete_event	461
icap_snooze	461
icap_list_alarms	461
XXIX. Image functions	463
GetImageSize	465
ImageArc	465
ImageChar	466
ImageCharUp	466
ImageColorAllocate	466
ImageColorDeAllocate	466
ImageColorAt	467
ImageColorClosest	467
ImageColorExact	467
ImageColorResolve	467
ImageGammaCorrect	468
ImageColorSet	468
ImageColorsForIndex	468
ImageColorsTotal	468
ImageColorTransparent	468
ImageCopy	469
ImageCopyResized	469
ImageCreate	469
ImageCreateFromGIF	470
ImageCreateFromJPEG	470
ImageCreateFromPNG	471
ImageCreateFromWBMP	471
ImageCreateFromString	472
ImageDashedLine	472
ImageDestroy	472
ImageFill	472
ImageFilledPolygon	472
ImageFilledRectangle	473
ImageFillToBorder	473
ImageFontHeight	473
ImageFontSize	473
ImageGIF	473

ImagePNG	474
ImageJPEG	475
ImageWBMP	475
ImageInterlace	476
ImageLine	476
ImageLoadFont	476
ImagePolygon	476
ImagePSBBox	477
ImagePSEncodeFont	477
ImagePSFreeFont	478
ImagePSLoadFont	478
ImagePsExtendFont	478
ImagePsSlantFont	478
ImagePSText	478
ImageRectangle	479
ImageSetPixel	479
ImageString	480
ImageStringUp	480
ImageSX	480
ImageSY	480
ImageTTFBox	480
ImageTTFTText	481
ImageTypes	482
read_exif_data	482
XXX. IMAP, POP3 and NNTP functions.	485
imap_append	487
imap_base64	487
imap_body	487
imap_check	488
imap_close	488
imap_createmailbox	488
imap_delete	489
imap_deletemailbox	490
imap_expunge	490
imap_fetchbody	490
imap_fetchstructure	491
imap_headerinfo	492
imap_header	494
imap_rfc822_parse_headers	494
imap_headers	494
imap_listmailbox	494
imap_getmailboxes	495
imap_listsubscribed	496
imap_getsubscribed	496
imap_mail_copy	496
imap_mail_move	497
imap_num_msg	497
imap_num_recent	497
imap_open	497
imap_ping	499
imap_renamemailbox	499
imap_reopen	499
imap_subscribe	499
imap_undelete	500
imap_unsubscribe	500
imap_qprint	500

imap_8bit.....	500
imap_binary.....	500
imap_scanmailbox.....	501
imap_mailboxmsginfo.....	501
imap_rfc822_write_address	502
imap_rfc822_parse_adrlist	502
imap_setflag_full.....	503
imap_clearflag_full.....	503
imap_sort.....	504
imap_fetchheader.....	504
imap_uid.....	505
imap_msgno.....	505
imap_search.....	505
imap_last_error.....	506
imap_errors.....	506
imap_alerts.....	507
imap_status.....	507
imap_utf7_decode	508
imap_utf7_encode	508
imap_utf8	508
imap_fetch_overview	508
imap_mime_header_decode	509
imap_mail_compose.....	510
imap_mail.....	510
XXXI. Informix functions	513
ifx_connect.....	515
ifx_pconnect.....	515
ifx_close.....	515
ifx_query.....	516
ifx_prepare	517
ifx_do	517
ifx_error.....	518
ifx_errormsg.....	518
ifx_affected_rows	518
ifx_getsqlca	519
ifx_fetch_row	520
ifx_htmltbl_result	520
ifx_fieldtypes	521
ifx_fieldproperties	521
ifx_num_fields	522
ifx_num_rows	522
ifx_free_result	522
ifx_create_char	522
ifx_free_char	523
ifx_update_char	523
ifx_get_char	523
ifx_create_blob	523
ifx_copy_blob	523
ifx_free_blob	524
ifx_get_blob	524
ifx_update_blob	524
ifx_blobinfile_mode	524
ifx_textasvarchar	524
ifx_byteasvarchar	525
ifx_nullformat.....	525
ifxus_create_slob.....	525

ifxus_free_slob	525
ifxus_close_slob	525
ifxus_open_slob	526
ifxus_tell_slob	526
ifxus_seek_slob	526
ifxus_read_slob	526
ifxus_write_slob	526
XXXII. InterBase functions	529
ibase_connect	531
ibase_pconnect	531
ibase_close	532
ibase_query	532
ibase_fetch_row	532
ibase_fetch_object	532
ibase_field_info	533
ibase_free_result	533
ibase_prepare	533
ibase_execute	534
ibase_trans	534
ibase_commit	534
ibase_rollback	534
ibase_free_query	535
ibase_timefmt	535
ibase_num_fields	535
ibaseerrmsg	536
XXXIII. Ingres II functions	537
ingres_connect	539
ingres_pconnect	539
ingres_close	539
ingres_query	540
ingres_num_rows	541
ingres_num_fields	541
ingres_field_name	541
ingres_field_type	542
ingres_field_nullable	542
ingres_field_length	542
ingres_field_precision	542
ingres_field_scale	543
ingres_fetch_array	543
ingres_fetch_row	544
ingres_fetch_object	544
ingres_rollback	545
ingres_commit	545
ingres_autocommit	545
XXXIV. LDAP functions	547
ldap_add	551
ldap_bind	551
ldap_close	551
ldap_compare	552
ldap_connect	553
ldap_count_entries	553
ldap_delete	553
ldap_dn2ufn	553
ldap_err2str	554
ldap_errno	554
ldap_error	555

ldap_explode_dn	555
ldap_first_attribute	555
ldap_first_entry	556
ldap_free_result	556
ldap_get_attributes	556
ldap_get_dn	557
ldap_get_entries	557
ldap_get_option	558
ldap_get_values	558
ldap_get_values_len	559
ldap_list	559
ldap_modify	560
ldap_mod_add	560
ldap_mod_del	560
ldap_mod_replace	561
ldap_next_attribute	561
ldap_next_entry	561
ldap_read	561
ldap_search	562
ldap_set_option	563
ldap_unbind	564
XXXV. Mail functions	565
mail	567
ezmlm_hash	568
XXXVI. Mathematical Functions	569
abs	571
acos	571
asin	571
atan	571
atan2	571
base_convert	572
bindec	572
ceil	572
cos	572
decbin	573
dechex	573
decoct	573
deg2rad	573
exp	573
floor	574
getrandmax	574
hexdec	574
lcg_value	574
log	575
log10	575
max	575
min	575
mt_rand	575
mt_srand	576
mt_getrandmax	576
number_format	576
octdec	577
pi	577
pow	577
rad2deg	577
rand	578

round	578
sin	578
sqrt	579
srand	579
tan	579
XXXVII. MCAL functions	581
mcal_open	583
mcal_popen	583
mcal_reopen	583
mcal_close	583
mcal_create_calendar	583
mcal_rename_calendar	584
mcal_delete_calendar	584
mcal_fetch_event	584
mcal_list_events	585
mcal_append_event	585
mcal_store_event	585
mcal_delete_event	585
mcal_snooze	586
mcal_list_alarms	586
mcal_event_init	586
mcal_event_set_category	586
mcal_event_set_title	586
mcal_event_set_description	587
mcal_event_set_start	587
mcal_event_set_end	587
mcal_event_set_alarm	587
mcal_event_set_class	588
mcal_is_leap_year	588
mcal_days_in_month	588
mcal_date_valid	588
mcal_time_valid	588
mcal_day_of_week	589
mcal_day_of_year	589
mcal_date_compare	589
mcal_next_recurrence	589
mcal_event_set_recur_none	589
mcal_event_set_recur_daily	590
mcal_event_set_recur_weekly	590
mcal_event_set_recur_monthly_mday	590
mcal_event_set_recur_monthly_wday	590
mcal_event_set_recur_yearly	590
mcal_fetch_current_stream_event	591
mcal_event_add_attribute	591
mcal_expunge	591
XXXVIII. Mcrypt Encryption Functions	593
mcrypt_get_cipher_name	597
mcrypt_get_block_size	597
mcrypt_get_key_size	597
mcrypt_create_iv	598
mcrypt_cbc	598
mcrypt_cfb	598
mcrypt_ecb	599
mcrypt_ofb	599
mcrypt_list_algorithms	600
mcrypt_list_modes	600

mcrypt_get_iv_size	601
mcrypt_encrypt.....	601
mcrypt_decrypt.....	602
mcrypt_module_open.....	602
mcrypt_generic_init	603
mcrypt_generic.....	603
mdecrypt_generic	603
mcrypt_generic_end.....	604
mcrypt_enc_self_test.....	604
mcrypt_enc_is_block_algorithm_mode.....	604
mcrypt_enc_is_block_algorithm.....	604
mcrypt_enc_is_block_mode.....	604
mcrypt_enc_get_block_size	605
mcrypt_enc_get_key_size	605
mcrypt_enc_get_supported_key_sizes.....	605
mcrypt_enc_get_iv_size	605
mcrypt_enc_get_algorithms_name	605
mcrypt_enc_get_modes_name	606
mcrypt_module_self_test	606
mcrypt_module_is_block_algorithm_mode.....	606
mcrypt_module_is_block_algorithm	606
mcrypt_module_is_block_mode	606
mcrypt_module_get_algo_block_size.....	607
mcrypt_module_get_algo_key_size	607
mcrypt_module_get_algo_supported_key_sizes.....	607
XXXIX. Mhash Functions	609
mhash_get_hash_name.....	611
mhash_get_block_size	611
mhash_count.....	611
mhash	612
mhash_keygen_s2k	612
XL. Microsoft SQL Server functions.....	613
mssql_close	615
mssql_connect	615
mssql_data_seek	615
mssql_fetch_array	615
mssql_fetch_field	616
mssql_fetch_object	616
mssql_fetch_row	616
mssql_field_length	617
mssql_field_name	617
mssql_field_seek	617
mssql_field_type	617
mssql_free_result	617
mssql_get_last_message	618
mssql_min_error_severity	618
mssql_min_message_severity	618
mssql_num_fields	618
mssql_num_rows	618
mssql_pconnect	618
mssql_query	619
mssql_result	619
mssql_select_db	619
XLI. Miscellaneous functions.....	621
connection_aborted	623
connection_status	623

connection_timeout	623
define	623
defined	624
die	624
eval	624
exit	625
get_browser	625
highlight_file	626
highlight_string	627
ignore_user_abort	628
iptcparse	628
leak	628
pack	628
show_source	629
sleep	630
uniqid	630
unpack	630
usleep	631
XLII. mSQL functions	633
mssql	635
mssql_affected_rows	635
mssql_close	635
mssql_connect	635
mssql_create_db	636
mssql_createdb	636
mssql_data_seek	636
mssql_dbname	636
mssql_drop_db	636
mssql_dropdb	637
mssql_error	637
mssql_fetch_array	637
mssql_fetch_field	637
mssql_fetch_object	638
mssql_fetch_row	638
mssql_fieldname	638
mssql_field_seek	639
mssql_fieldtable	639
mssql_fieldtype	639
mssql_fieldflags	639
mssql_fieldlen	639
mssql_free_result	640
mssql_freeresult	640
mssql_list_fields	640
mssql_listfields	640
mssql_list_dbs	640
mssql_listdbs	641
mssql_list_tables	641
mssql_listtables	641
mssql_num_fields	641
mssql_num_rows	641
mssql_numfields	641
mssql_numrows	642
mssql_pconnect	642
mssql_query	642
mssql_regcase	642
mssql_result	643

msql_select_db	643
msql_selectdb	643
msql_tablename	643
XLIII. MySQL functions	645
mysql_affected_rows	647
mysql_change_user	647
mysql_close	647
mysql_connect	648
mysql_create_db	648
mysql_data_seek	649
mysql_db_name	650
mysql_db_query	650
mysql_drop_db	650
mysql_errno	651
mysql_error	651
mysql_fetch_array	652
mysql_fetch_assoc	652
mysql_fetch_field	653
mysql_fetch_lengths	654
mysql_fetch_object	654
mysql_fetch_row	655
mysql_field_flags	655
mysql_field_name	655
mysql_field_len	656
mysql_field_seek	656
mysql_field_table	657
mysql_field_type	657
mysql_free_result	657
mysql_insert_id	658
mysql_list_dbs	658
mysql_list_fields	659
mysql_list_tables	659
mysql_num_fields	660
mysql_num_rows	660
mysql_pconnect	660
mysql_query	661
mysql_result	662
mysql_select_db	662
mysql_tablename	662
XLIV. Network Functions	665
checkdnsrr	667
closelog	667
debugger_off	667
debugger_on	667
define_syslog_variables	667
fsockopen	668
gethostbyaddr	669
gethostbyname	669
gethostbynamel	669
getmxrr	669
getprotobynumber	669
getservbyname	670
getservbyport	670
ip2long	670
long2ip	671

openlog.....	671
pfsockopen.....	672
socket_get_status.....	672
socket_set_blocking.....	672
socket_set_timeout.....	673
syslog.....	673
XLV. Unified ODBC functions	675
odbc_autocommit.....	677
odbc_binmode.....	677
odbc_close.....	677
odbc_close_all.....	678
odbc_commit.....	678
odbc_connect.....	678
odbc_cursor.....	679
odbc_do	679
odbc_error	679
odbc_errormsg.....	679
odbc_exec.....	680
odbc_execute	680
odbc_fetch_into.....	680
odbc_fetch_row.....	680
odbc_field_name.....	681
odbc_field_num.....	681
odbc_field_type.....	681
odbc_field_len.....	681
odbc_field_precision.....	681
odbc_field_scale.....	682
odbc_free_result.....	682
odbc_longreadlen.....	682
odbc_num_fields.....	682
odbc_pconnect.....	683
odbc_prepare.....	683
odbc_num_rows.....	683
odbc_result	683
odbc_result_all.....	684
odbc_rollback.....	684
odbc_setopt.....	684
odbc_tables.....	685
odbc_tableprivileges.....	686
odbc_columns.....	686
odbc_columnprivileges.....	687
odbc_gettypeinfo	688
odbc_primarykeys	688
odbc_foreignkeys	689
odbc_procedures	689
odbc_procedurecolumns	690
odbc_specialcolumns	691
odbc_statistics	691
XLVI. Oracle 8 functions	693
OCIDefineByName	695
OCIBindByName	695
OCILogon	696
OCIPLogon	698
OCINLogon	698
OCILogOff	700
OCIEexecute	700

OCICommit	701
OCIRollback	701
OCINewDescriptor	701
OCIRowCount	702
OCINumCols	703
OCIResult	703
OCIFetch	704
OCIFetchInto	704
OCIFetchStatement	704
OCIColumnIsNULL	705
OCIColumnName	705
OCIColumnSize	706
OCIColumnType	707
OCIServerVersion	707
OCISatementType	708
OCINewCursor	709
OCIFreeStatement	710
OCIFreeCursor	710
OCIFreeDesc	710
OCIParse	710
OCIError	711
OCIInternalDebug	711
XLVII. OpenSSL functions	713
openssl_free_key	715
openssl_get_privatekey	715
openssl_get_publickey	715
openssl_open	715
openssl_seal	716
openssl_sign	716
openssl_verify	717
XLVIII. Oracle functions	719
Ora_Bind	721
Ora_Close	721
Ora_ColumnName	721
Ora_ColumnSize	721
Ora_ColumnType	722
Ora_Commit	722
Ora_CommitOff	722
Ora_CommitOn	723
Ora_Do	723
Ora_Error	723
Ora_ErrorCode	723
Ora_Exec	724
Ora_Fetch	724
Ora_Fetch_Into	724
Ora_GetColumn	725
Ora_Logoff	725
Ora_Logon	725
Ora_pLogon	725
Ora_Numcols	726
Ora_Numrows	726
Ora_Open	726
Ora_Parse	726
Ora_Rollback	727
XLIX. Ovrimos SQL functions	729
ovrimos_connect	731

ovrimos_close.....	731
ovrimos_close_all.....	731
ovrimos_longreadlen.....	731
ovrimos_prepare.....	732
ovrimos_execute.....	732
ovrimos_cursor.....	733
ovrimos_exec.....	733
ovrimos_fetch_into.....	733
ovrimos_fetch_row.....	734
ovrimos_result.....	735
ovrimos_result_all.....	735
ovrimos_num_rows.....	736
ovrimos_num_fields.....	736
ovrimos_field_name.....	737
ovrimos_field_type.....	737
ovrimos_field_len.....	737
ovrimos_field_num.....	737
ovrimos_free_result.....	737
ovrimos_commit.....	738
ovrimos_rollback.....	738
L. Output Control Functions.....	739
flush.....	741
ob_start.....	741
ob_get_contents.....	742
ob_get_length.....	742
ob_end_flush.....	742
ob_end_clean.....	742
ob_implicit_flush.....	742
LI. PDF functions.....	745
pdf_set_info.....	751
pdf_open.....	751
pdf_close.....	751
pdf_begin_page.....	752
pdf_end_page.....	752
pdf_show.....	752
pdf_show_boxed.....	752
pdf_show_xy.....	753
pdf_set_font.....	753
pdf_set_leading.....	753
pdf_set_parameter.....	753
pdf_get_parameter.....	754
pdf_set_value.....	754
pdf_get_value.....	754
pdf_get_image_height.....	754
pdf_get_image_width.....	755
pdf_set_text_rendering.....	755
pdf_set_horiz_scaling.....	755
pdf_set_text_rise.....	755
pdf_set_text_matrix.....	755
pdf_set_text_pos.....	756
pdf_set_char_spacing.....	756
pdf_set_word_spacing.....	756
pdf_skew.....	756
pdf_continue_text.....	756
pdf_stringwidth.....	757
pdf_save.....	757

pdf_restore	757
pdf_translate	757
pdf_scale	758
pdf_rotate	758
pdf_setflat	758
pdf_setlinejoin	759
pdf_setlinecap	759
pdf_setmiterlimit	759
pdf_setlinewidth	759
pdf_setdash	759
pdf_moveto	760
pdf_curveto	760
pdf_lineto	760
pdf_circle	760
pdf_arc	760
pdf_rect	761
pdf_closepath	761
pdf_stroke	761
pdf_closepath_stroke	761
pdf_fill	762
pdf_fill_stroke	762
pdf_closepath_fill_stroke	762
pdf_endpath	762
pdf_clip	762
pdf_setgray_fill	763
pdf_setgray_stroke	763
pdf_setgray	763
pdf_setrgbcolor_fill	763
pdf_setrgbcolor_stroke	763
pdf_setrgbcolor	764
pdf_add_outline	764
pdf_set_transition	764
pdf_set_duration	765
pdf_open_gif	765
pdf_open_png	765
pdf_open_image_file	766
pdf_open_memory_image	766
pdf_open_jpeg	767
pdf_open_tiff	767
pdf_close_image	767
pdf_place_image	768
pdf_put_image	768
pdf_execute_image	768
pdf_add_annotation	769
pdf_set_border_style	769
pdf_set_border_color	769
pdf_set_border_dash	769
LII. Verisign Payflow Pro functions	771
pfpro_init	773
pfpro_cleanup	773
pfpro_process	773
pfpro_process_raw	774
pfpro_version	775
LIII. PHP options & information	777
assert	779
assert_options	779

extension_loaded	779
dl	780
getenv	780
get_cfg_var	780
get_current_user	780
get_magic_quotes_gpc	781
get_magic_quotes_runtime	781
getlastmod	781
getmyinode	781
getmypid	782
getmyuid	782
getusage	782
ini_alter	783
ini_get	783
ini_restore	783
ini_set	783
phpcredits	784
phpinfo	785
phpversion	785
php_logo_guid	786
php_sapi_name	786
php_uname	786
putenv	787
set_magic_quotes_runtime	787
set_time_limit	787
zend_logo_guid	787
get_loaded_extensions	788
get_extension_funcs	788
get_required_files	789
get_included_files	790
LIV. POSIX functions	791
posix_kill	793
posix_getpid	793
posix_getppid	793
posix_getuid	793
posix_geteuid	793
posix_getgid	793
posix_getegid	794
posix_setuid	794
posix_setgid	794
posix_getgroups	794
posix_getlogin	795
posix_getpgrp	795
posix_setsid	795
posix_setpgid	795
posix_getpgid	795
posix_getsid	796
posix_uname	796
posix_times	796
posix_ctermid	797
posix_ttyname	797
posix_isatty	797
posix_getcwd	797
posix_mkfifo	797
posix_getgrnam	797
posix_getgrgid	798

posix_getpwnam	798
posix_getpwuid	799
posix_getrlimit	799
LV. PostgreSQL functions	801
pg_close	803
pg_cmtuples	803
pg_connect	803
pg_dbname	804
pg_end_copy	804
pg_errormessage	804
pg_exec	804
pg_fetch_array	805
pg_fetch_object	805
pg_fetch_row	807
pg_fieldisnull	807
pg_fieldname	808
pg_fieldnum	808
pg_fieldprtlen	808
pg_fieldsize	808
pg_fieldtype	808
pg_freeresult	809
pg_getlastoid	809
pg_host	809
pg_loclose	809
pg_locreate	809
pg_loexport	810
pg_loimport	810
pg_loopen	810
pg_loread	810
pg_loreadall	810
pg_lounlink	811
pg_lowrite	811
pg_numfields	811
pg_numrows	811
pg_options	811
pg_pconnect	812
pg_port	812
pg_put_line	812
pg_result	813
pg_set_client_encoding	813
pg_client_encoding	813
pg_trace	814
pg_tty	814
pg_untrace	814
LVI. Program Execution functions	815
escapeshellarg	817
escapeshellcmd	817
exec	817
passthru	818
system	818
LVII. Pspell Functions	819
pspell_add_to_personal	821
pspell_add_to_session	821
pspell_check	821
pspell_clear_session	821
pspell_config_create	822

pspell_config_ignore	823
pspell_config_mode	823
pspell_config_personal	824
pspell_config_repl	824
pspell_config_runtogther	824
pspell_config_save_repl	825
pspell_new	825
pspell_new_config	826
pspell_new_personal	826
pspell_save_wordlist	827
pspell_store_replacement	828
pspell_suggest	828
LVIII. GNU Readline	829
readline	831
readline_add_history	831
readline_clear_history	831
readline_completion_function	831
readline_info	831
readline_list_history	832
readline_read_history	832
readline_write_history	832
LIX. GNU Recode functions	833
recode_string	835
recode	835
recode_file	835
LX. Regular Expression Functions (Perl-Compatible)	837
preg_match	839
preg_match_all	839
preg_replace	841
preg_split	843
preg_quote	843
preg_grep	844
Pattern Modifiers	844
Pattern Syntax	846
LXI. Regular Expression Functions (POSIX Extended)	869
ereg	871
ereg_replace	871
eregi	872
eregi_replace	872
split	872
spli	873
sql_regcase	873
LXII. Satellite CORBA client extension	875
OrbitObject	877
OrbitEnum	877
OrbitStruct	878
satellite_caught_exception	878
satellite_exception_id	879
satellite_exception_value	879
LXIII. Semaphore and Shared Memory Functions	881
sem_get	883
sem_acquire	883
sem_release	883
shm_attach	883
shm_detach	884
shm_remove	884

shm_put_var	884
shm_get_var	884
shm_remove_var	885
LXIV. SESAM database functions	887
sesam_connect	891
sesam_disconnect	891
sesam_settransaction	892
sesam_commit	892
sesam_rollback	893
sesam_execimm	893
sesam_query	894
sesam_num_fields	895
sesam_field_name	895
sesam_diagnostic	896
sesam_fetch_result	897
sesam_affected_rows	898
sesam_errormsg	899
sesam_field_array	899
sesam_fetch_row	901
sesam_fetch_array	903
sesam_seek_row	904
sesam_free_result	904
LXV. Session handling functions	907
session_start	911
session_destroy	911
session_name	911
session_module_name	911
session_save_path	912
session_id	912
session_register	912
session_unregister	913
session_unset	913
session_is_registered	913
session_get_cookie_params	913
session_set_cookie_params	914
session_decode	914
session_encode	914
session_set_save_handler	914
session_cache_limiter	916
LXVI. Shared Memory Functions	919
shmop_open	921
shmop_read	921
shmop_write	921
shmop_size	922
shmop_delete	922
shmop_close	923
LXVII. Shockwave Flash functions	925
swf_openfile	927
swf_closefile	927
swf_labelframe	928
swf_showframe	928
swf_setframe	928
swf_getframe	929
swf_mulcolor	929
swf_adddcolor	929
swf_placeobject	929

swf_modifyobject.....	930
swf_removeobject	930
swf_nextid	930
swf_startdoaction	930
swf_actiongotoframe	931
swf_actiongeturl	931
swf_actionnextframe	931
swf_actionprevframe	931
swf_actionplay	931
swf_actionstop	931
swf_actiontogglequality	932
swf_actionwaitforframe	932
swf_actionsettarget	932
swf_actiongotolabel	932
swf_enddoaction	932
swf_defineline	933
swf_definerect	933
swf_definepoly	933
swf_startshape	933
swf_shapelinesolid	933
swf_shapefilloff	934
swf_shapefillsolid	934
swf_shapefillbitmapclip	934
swf_shapefillbitmaptile	934
swf_shapemoveto	934
swf_shapelineto	935
swf_shapecurveto	935
swf_shapecurveto3	935
swf_shapearc	935
swf_endshape	935
swf_definefont	936
swf_setfont	936
swf_fontsize	936
swf_fontslant	936
swf_fontracking	936
swf_getfontinfo	937
swf_definetext	937
swf_textwidth	937
swf_definebitmap	937
swf_getbitmapinfo	938
swf_startsymbol	938
swf_endsymbol	938
swf_startbutton	938
swf_addbuttonrecord	938
swf_oncondition	939
swf_endbutton	940
swf_viewport	940
swf_ortho	940
swf_ortho2	940
swf_perspective	940
swf_polarview	941
swf_lookat	941
swf_pushmatrix	941
swf_popmatrix	941
swf_scale	942
swf_translate	942

swf_rotate	942
swf_posround	942
LXVIII. SNMP functions.....	943
snmpget	945
snmpset	945
snmpwalk	945
snmpwalkoid	946
snmp_get_quick_print.....	946
snmp_set_quick_print	946
LXIX. Socket functions	949
accept_connect	953
bind	953
close	953
connect	953
listen	954
read	954
socket.....	954
strerror	955
write	955
LXX. String functions.....	957
AddCSlashes	959
AddSlashes	959
bin2hex	959
Chop	959
Chr	960
chunk_split	960
convert_cyr_string.....	960
count_chars.....	961
crc32	961
crypt.....	961
echo	962
explode	963
get_html_translation_table	963
get_meta_tags	964
hebrev	964
hebreve	964
htmlentities	965
htmlspecialchars	965
implode	966
join	966
levenshtein	966
localeconv	967
ltrim	969
md5	969
Metaphone	969
nl2br	970
Ord	970
parse_str	970
print	971
printf	971
quoted_printable_decode	971
quotemeta	971
rtrim	971
sscanf	972
setlocale	972
similar_text	973

soundex.....	973
sprintf	974
strncasecmp	975
strcasecmp	975
strchr	975
strcmp	975
strcoll	976
strcspn	976
strip_tags	976
stripcslashes	977
stripslashes	977
stristr	977
strlen	977
strnatcasecmp	977
strnatcasecmp	978
strncmp	979
str_pad	979
strpos	979
strrchr	980
str_repeat	980
strrev	981
strrpos	981
strrspan	982
strrstr	982
strtok	982
strtolower	983
strtoupper	983
str_replace	984
strtr	984
substr	985
substr_count	986
substr_replace	986
trim	987
ucfirst	987
ucwords	987
wordwrap	988
LXXI. Sybase functions.....	991
sybase_affected_rows	993
sybase_close	993
sybase_connect	993
sybase_data_seek	993
sybase_fetch_array	994
sybase_fetch_field	994
sybase_fetch_object	994
sybase_fetch_row	995
sybase_field_seek	995
sybase_free_result	995
sybase_get_last_message	996
sybase_min_client_severity	996
sybase_min_error_severity	996
sybase_min_message_severity	996
sybase_min_server_severity	996
sybase_num_fields	997
sybase_num_rows	997
sybase_pconnect	997
sybase_query	998

sybase_result	998
sybase_select_db	998
LXXII. URL Functions	999
base64_decode	1001
base64_encode	1001
parse_url	1001
rawurldecode	1001
rawurlencode	1001
urldecode	1002
urlencode	1002
LXXIII. Variable Functions	1005
doubleval	1007
empty	1007
gettype	1007
intval	1008
is_array	1008
is_bool	1008
is_double	1008
is_float	1009
is_int	1009
is_integer	1009
is_long	1009
is_numeric	1009
is_object	1010
is_real	1010
is_resource	1010
is_string	1010
isset	1011
print_r	1011
serialize	1011
settype	1012
strval	1012
unserialize	1013
unset	1013
var_dump	1015
LXXIV. WDDX functions	1017
wddx_serialize_value	1019
wddx_serialize_vars	1019
wddx_packet_start	1019
wddx_packet_end	1020
wddx_add_vars	1020
wddx_deserialize	1020
LXXV. XML parser functions	1021
xml_parser_create	1029
xml_set_object	1029
xml_set_element_handler	1030
xml_set_character_data_handler	1030
xml_set_processing_instruction_handler	1031
xml_set_default_handler	1032
xml_set_unparsed_entity_decl_handler	1032
xml_set_notation_decl_handler	1033
xml_set_external_entity_ref_handler	1034
xml_parse	1035
xml_get_error_code	1035
xml_error_string	1036
xml_get_current_line_number	1036

xml_get_current_column_number	1036
xml_get_current_byte_index	1036
xml_parse_into_struct	1037
xml_parser_free	1040
xml_parser_set_option	1040
xml_parser_get_option	1041
utf8_decode	1041
utf8_encode	1041
LXXVI. XSLT functions	1043
xslt_closelog	1045
xslt_create	1045
xslt_errno	1045
xslt_error	1045
xslt_fetch_result	1045
xslt_free	1046
xslt_openlog	1046
xslt_output_begintransform	1046
xslt_output_endtransform	1047
xslt_process	1047
xslt_run	1048
xslt_set_sax_handler	1048
xslt_transform	1048
LXXVII. YAZ functions	1051
yaz_addinfo	1053
yaz_close	1053
yaz_connect	1053
yaz_errno	1053
yaz_error	1053
yaz_hits	1054
yaz_element	1054
yaz_range	1054
yaz_record	1054
yaz_search	1054
yaz_syntax	1055
yaz_wait	1056
LXXVIII. YP/NIS Functions	1057
yp_get_default_domain	1059
yp_order	1059
yp_master	1059
yp_match	1060
yp_first	1060
yp_next	1060
LXXIX. Zlib Compression Functions	1063
gzclose	1065
gzeof	1065
gzfile	1065
gzgetc	1065
gzgets	1065
gzgetss	1066
gzopen	1066
gzpassthru	1066
gzputs	1067
gzread	1067
gzrewind	1067
gzseek	1068
gztell	1068

gzwrite	1068
readgzfile	1068
gzcompress	1069
gzuncompress	1069
V. Appendixes	1071
A. Migrating from older versions of PHP	1071
Migrating from PHP 3 to PHP 4	1073
Running PHP 3 and PHP 4 concurrently	1073
Migrating Configuration Files	1073
Global Configuration File	1073
Apache Configuration Files	1073
Migrating from PHP/FI 2.0 to PHP 3.0	1074
About the incompatibilities in 3.0	1074
Start/end tags	1074
if..endif syntax	1075
while syntax	1075
Expression types	1076
Error messages have changed	1076
Short-circuited boolean evaluation	1076
Function true/false return values	1076
Other incompatibilities	1077
B. Migrating from PHP 3.0 to PHP 4.0	1079
What has changed in PHP 4.0	1081
Parser behavior	1081
Error reporting	1081
Configuration changes	1081
Additional warning messages	1081
Initializers	1082
empty("0")	1082
Missing functions	1082
Functions missing due to conceptual changes	1082
Deprecate functions and extensions	1082
Changed status for unset()	1082
PHP 3.0 extension	1083
Variable substitution in strings	1083
Cookies	1083
C. PHP development	1085
Adding functions to PHP 3	1087
Function Prototype	1087
Function Arguments	1087
Variable Function Arguments	1087
Using the Function Arguments	1087
Memory Management in Functions	1088
Setting Variables in the Symbol Table	1089
Returning simple values	1090
Returning complex values	1091
Using the resource list	1092
Using the persistent resource table	1093
Adding runtime configuration directives	1094
Calling User Functions	1094
HashTable *function_table	1094
pval *object	1094
pval *function_name	1094
pval *retval	1095
int param_count	1095

pval *params[]	1095
Reporting Errors	1095
E_NOTICE	1095
E_WARNING	1095
E_ERROR	1095
E_PARSE	1095
E_CORE_ERROR	1095
E_CORE_WARNING	1095
E_COMPILE_ERROR	1096
E_COMPILE_WARNING	1096
E_USER_ERROR	1096
E_USER_WARNING	1096
E_USER_NOTICE	1096
D. The PHP Debugger	1097
About the debugger	1099
Using the Debugger	1099
Debugger Protocol	1099

Preface

PHP, which stands for "PHP: Hypertext Preprocessor", is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with some unique 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 XML using the DocBook XML DTD (<http://www.nwalsh.com/docbook/xml/>), 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 is maintained by Stig Sæther Bakken (<mailto:stig@php.net>).

You can download the manual in various languages and formats, including PDF, plain text, plain HTML, WinHelp, and RTF from <http://www.php.net/docs.php>.

Daily HTML snapshots of the manual, including translations, can be found at <http://snaps.php.net/manual/>.

You can find more information about downloading the XML source code of this documentation at <http://cvs.php.net/>. The documentation is stored in the `phpdoc` module.

Part I. Getting Started

Chapter 1. Introduction

What is PHP?

PHP (officially "PHP: Hypertext Preprocessor") 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 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	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (read-only)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

PHP also has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, HTTP and countless others. 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@php.net>). 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 PHP 3 and a lot of it was completely rewritten.

The latest version (PHP 4) uses the Zend (<http://www.zend.com/>) scripting engine to deliver higher performance, supports an even wider array of third-party libraries and extensions, and runs as a native server module with all of the popular web servers.

Today (1/2001) PHP 3 or PHP 4 now ships with a number of commercial products such as Red Hat's Stronghold web server. A conservative estimate based on an extrapolation from numbers provided by Netcraft (<http://www.netcraft.com/>) (see also Netcraft Web Server Survey (<http://www.netcraft.com/survey/>)) would be that PHP is in use on over 5,100,000 sites around the world. To put that in perspective, that is slightly more sites than run Microsoft's IIS server on the Internet (5.03 million).

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/>. We recommend you to choose mirror (<http://www.php.net/mirrors.php>) nearest to you for downloading the distributions.

Installation on UNIX systems

This section will guide you through the general configuration and installation of PHP on unix systems. Be sure to investigate any sections specific to your platform or web server before you begin the process.

Prerequisite knowledge and software:

- Basic UNIX skills (being able to operate "make" and a C compiler, if compiling)
- An ANSI C compiler (if compiling)
- flex (for compiling)
- bison (for compiling)
- A web server
- Any module specific components (such as gd, pdf libs, etc.)

There are several ways to install PHP for the Unix platform, either with a compile and configure process, or through various pre-packaged methods. This documentation is mainly focused around the process of compiling and configuring PHP.

The initial PHP setup and configuration process is controlled by the use of the commandline options of the `configure` script. This page outlines the usage of the most common options, but there are many others to play with. Check out the [Complete list of configure options](#) for an exhaustive rundown. There are several ways to install PHP:

- As an [Apache module](#)
- As an [fhttpd module](#)
- For use with [AOLServer](#), [NSAPI](#), [phttpd](#), [Pi3Web](#), [Roxen](#), [thttpd](#), or [Zeus](#).
- As a [CGI executable](#)

Apache Module Quick Reference

PHP can be compiled in a number of different ways, but one of the most popular is as an Apache module. The following is a quick installation overview.

Example 2-1. Quick Installation Instructions for PHP 4 (Apache Module Version)

1. `gunzip apache_1.3.x.tar.gz`
2. `tar xvf apache_1.3.x.tar`
3. `gunzip php-x.x.x.tar.gz`
4. `tar xvf php-x.x.x.tar`
5. `cd apache_1.3.x`
6. `./configure --prefix=/www`
7. `cd ../php-x.x.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 --activate-module=src/modules/php4/libphp4.a`
13. `make`

```

14. make install
15. cd ..../php-x.x.x
16. cp php.ini-dist /usr/local/lib/php.ini
17. Edit your httpd.conf or srm.conf file and add:
    AddType application/x-httpd-php .php

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

```

Building

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

Linux installs

This section contains notes and hints specific to installing PHP on Linux distributions.

Using Packages

Many Linux distributions have some sort of package installation, such as RPM. This can assist in setting up a standard configuration, but if you need to have a different set of features (such as a secure server, or a different database driver), you may need to build php and/or your webserver. If you are unfamiliar with building and compiling your own software, it is worth checking to see whether somebody has already built a packaged version of PHP with the features you need.

HP-UX installs

This section contains notes and hints specific to installing PHP on HP-UX systems.

Example 2-2. Installation Instructions for HP-UX 10

```

From: paul_mckay@clearwater-it.co.uk
04-Jan-2001 09:49
(These tips are for PHP 4.0.4 and Apache v1.3.9)

```

So you want to install PHP and Apache on a HP-UX 10.20 box?

1. You need gzip, download a binary distribution from
<http://hpxx.connect.org.uk/ftp/hpxx/Gnu/gzip-1.2.4a/gzip-1.2.4a-sd-10.20.depot.Z>
uncompress the file and install using swinstall
2. You need gcc, download a binary distribution from
<http://gatekeep.cs.utah.edu/ftp/hpxx/Gnu/gcc-2.95.2/gcc-2.95.2-sd-10.20.depot.gz>
gunzip this file and install gcc using swinstall.
3. You need the GNU binutils, you can download a binary distribution from
<http://hpxx.connect.org.uk/ftp/hpxx/Gnu/binutils-2.9.1/binutils-2.9.1-sd-10.20.depot.gz>
gunzip and install using swinstall.
4. You now need bison, you can download a binary distribution from
<http://hpxx.connect.org.uk/ftp/hpxx/Gnu/bison-1.28/bison-1.28-sd-10.20.depot.gz>
install as above.
5. You now need flex, you need to download the source from one of the

<http://www.gnu.org> mirrors. It is in the non-gnu directory of the ftp site. Download the file, gunzip, then tar -xvf it. Go into the newly created flex directory and do a ./configure, then a make, and then a make install

If you have errors here, it's probably because gcc etc. are not in your PATH so add them to your PATH.

Right, now into the hard stuff.

6. Download the PHP and apache sources.
7. gunzip and tar -xvf them.

We need to hack a couple of files so that they can compile ok.

8. Firstly the configure file needs to be hacked because it seems to lose track of the fact that you are a hpx machine, there will be a better way of doing this but a cheap and cheerful hack is to put
 lt_target=hpx10.20
 on line 47286 of the configure script.

9. Next, the Apache GuessOS file needs to be hacked. Under
 apache_1.3.9/src/helpers change line 89 from
 "echo "hp\${HPUXMACH}-hpx\${HPUXVER}"; exit 0"
 to:
 "echo "hp\${HPUXMACH}-hp-hpx\${HPUXVER}"; exit 0"

10. You cannot install PHP as a shared object under HP-UX so you must compile it as a static, just follow the instructions at the Apache page.

11. PHP and apache should have compiled OK, but Apache won't start. you need to create a new user for Apache, eg www, or apache. You then change lines 252 and 253 of the conf/httpd.conf in Apache so that instead of
 User nobody
 Group nogroup
 you have something like
 User www
 Group sys

This is because you can't run Apache as nobody under hp-ux.
 Apache and PHP should then work.

Hope this helps somebody,
 Paul McKay.

Solaris installs

This section contains notes and hints specific to installing PHP on Solaris systems.

Required software

Solaris installs often lack C compilers and their related tools. The required software is as follows:

- gcc (recommended, other C compilers may work)
- make
- flex
- bison

- m4
- autoconf
- automake
- perl
- gzip
- tar

In addition, you will need to install (and possibly compile) any additional software specific to your configuration (such as Oracle or MySQL).

Using Packages

You can simplify the Solaris install process by using pkgadd to install most of your needed components.

Mac OS X installs

This section contains notes and hints specific to installing PHP on Mac OS X.

Using Packages

There are a few pre-packaged and pre-compiled versions of PHP for Mac OS X. This can help in setting up a standard configuration, but if you need to have a different set of features (such as a secure server, or a different database driver), you may need to build PHP and/or your web server yourself. If you are unfamiliar with building and compiling your own software, it's worth checking whether somebody has already built a packaged version of PHP with the features you need. Lightyear Design (<http://homepage.mac.com/LightyearDesign/MacOSX/Packages/>) offers a pre-built version of PHP for OS X, as does Tenon Intersystems (<http://www.tenon.com/products/webten/>).

Compiling for OS X server

There are two slightly different versions of Mac OS X, client and server. The following is for OS X Server.

Example 2-3. Mac OS X server install

1. Get the latest distributions of Apache and PHP
2. Untar them, and run the configure program on Apache like so.


```
./configure -exec-prefix=/usr \
-localstatedir=/var \
-mandir=/usr/share/man \
-libexecdir=/System/Library/Apache/Modules \
-iiconsdir=/System/Library/Apache/Icons \
-includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
-enable-shared=max \
-enable-module=most \
-target=apache
```
4. You may also want to add this line:


```
setenv OPTIM=-O2
```

If you want the compiler to do some optimization.
5. Next, go to the PHP 4 source directory and configure it.


```
./configure -prefix=/usr \
-sysconfdir=/etc \
-localstatedir=/var \
-mandir=/usr/share/man \
```

```

-with-xml \
-with-apache=/src/apache_1.3.12

If you have any other addiitons (MySQL, GD, etc.), be sure to add
them here. For the -with-apache string, put in the path to your
apache source directory, for example "/src/apache_1.3.12".
6. make
7. make install
    This will add a directory to your Apache source directory under
    src/modules/php4.

8. Now, reconfigure Apache to build in PHP 4.
    ./configure -exec-prefix=/usr \
    -localstatedir=/var \
    -mandir=/usr/share/man \
    -libexecdir=/System/Library/Apache/Modules \
    -iconsdir=/System/Library/Apache/Icons \
    -includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
    -enable-shared=max \
    -enable-module=most \
    -target=apache \
    -activate-module=src/modules/php4/libphp4.a

```

You may get a message telling you that libmodphp4.a is out of date.
 If so, go to the src/modules/php4 directory inside your apache
 source directory and run this command:

```
ranlib libmodphp4.a
```

Then go back to the root of the apache source directory and run the
 above configure command again. That'll bring the link table up to
 date.

```

9. make
10. make install
11. copy and rename the php.ini-dist file to your "bin" directory from your
    PHP 4 source directory:
    cp php.ini-dist /usr/local/bin/php.ini
    or (if your don't have a local directory)
    cp php.ini-dist /usr/bin/php.ini

```

Other examples for Mac OS X client (<http://www.stepwise.com/Articles/Workbench/Apache-1.3.14-MacOSX.html>) and
 Mac OS X server (<http://www.stepwise.com/Articles/Workbench/Apache-1.3.14-MacOSX.html>) are available at Stepwise
 (<http://www.stepwise.com/>).

Complete list of configure options

Note: These are only used at compile time. If you want to alter PHP's runtime configuration, please see the chapter on Configuration.

The following is a complete list of options supported by the PHP 3 and PHP 4 configure scripts, used when compiling in Unix-like environments. Some are available in PHP 3, some in PHP 4, and some in both, as noted. There are many options the names of which have changed between PHP 3 and PHP 4, but which accomplish the same things. These entries are

cross-referenced to each other, so if you have a problem getting your PHP 3-era configuration options to work, check here to see whether the names have changed.

- [Database](#)
- [Ecommerce](#)
- [Graphics](#)
- [Miscellaneous](#)
- [Networking](#)
- [PHP Behaviour](#)
- [Server](#)
- [Text and language](#)
- [XML](#)

Database

`-with-adabas[=DIR]`

PHP 3, PHP 4: Include Adabas D support. DIR is the Adabas base install directory, defaults to /usr/local.
[Adabas home page](http://www.adabas.com/) (<http://www.adabas.com/>)

`-enable-dba=shared`

PHP 3: Option not available in PHP 3
 PHP 4: Build DBA as a shared module

`-enable-dbase`

PHP 3: Option not available; use [`-with-dbase`](#) instead.
 PHP 4: Enable the bundled dbase library. No external libraries are required.

`-with-dbase`

PHP 3: Include the bundled dbase library. No external libraries are required.
 PHP 4: Option not available; use [`-enable-dbase`](#) instead.

`-with-db2[=DIR]`

PHP 3, PHP 4: Include Berkeley DB2 support

`-with-db3[=DIR]`

PHP 3: Option not available in PHP 3
 PHP 4: Include Berkeley DB3 support

`-with-dbm[=DIR]`

PHP 3, PHP 4: Include DBM support

`-with-dbmaker[=DIR]`

PHP 3: Option not available in PHP 3

PHP 4: Include DBMaker support. DIR is the DBMaker base install directory, defaults to where the latest version of DBMaker is installed (such as /home/dbmaker/3.6).

-with-empress[=DIR]

PHP 3, PHP 4: Include Empress support. DIR is the Empress base install directory, defaults to \$EMPRESSPATH
-enable-filepro

PHP 3: Option not available; use [-with-filepro](#) instead.

PHP 4: Enable the bundled read-only filePro support. No external libraries are required.

-with-filepro

PHP 3: Include the bundled read-only filePro support. No external libraries are required.

PHP 4: Option not available; use [-enable-filepro](#) instead.

-with-gdbm[=DIR]

PHP 3, PHP 4: Include GDBM support

-with-hyperwave

PHP 3, PHP 4: Include Hyperwave support

-with-ibm-db2[=DIR]

PHP 3, PHP 4: Include IBM DB2 support. DIR is the DB2 base install directory, defaults to
 /home/db2inst1/sqllib.

IBM DB2 home page (<http://www.ibm.com/db2/>)

-with-informix[=DIR]

PHP 3, PHP 4: Include Informix support. DIR is the Informix base install directory, defaults to nothing.

-with-ingres[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Ingres II support. DIR is the Ingres base directory (default /II/ingres)

-with-interbase[=DIR]

PHP 3, PHP 4: Include InterBase support. DIR is the InterBase base install directory, which defaults to
 /usr/interbase.

[Interbase functions](#)

Interbase home page (<http://www.interbase.com/>)

-with-ldap[=DIR]

PHP 3: Include LDAP support. DIR is the LDAP base install directory. Defaults to /usr and /usr/local

PHP 4: Include LDAP support. DIR is the LDAP base install directory.

This provides 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 (<http://www.faqs.org/rfcs/rfc1777.html>) and RFC1778 (<http://www.faqs.org/rfcs/rfc1778.html>).

-with-msql[=DIR]

PHP 3, PHP 4: 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/>)

-with-mysql[=DIR]

PHP 3: Include MySQL support. DIR is the MySQL base install directory, defaults to searching through a number of common places for the MySQL files.

PHP 4: Include MySQL support. DIR is the MySQL base directory. If unspecified, the bundled MySQL library will be used. This option is turned on by default.

See also [MySQL Configuration Directives](#) in the [configuration file](#).

MySQL home page (<http://www.mysql.com/>)

-with-ndbm[=DIR]

PHP 3, PHP 4: Include NDBM support

-with-oci8[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Oracle-oci8 support. Default DIR is ORACLE_HOME.

-with-oracle[=DIR]

PHP 3: Include Oracle database support. DIR is Oracle's home directory, defaults to \$ORACLE_HOME.

PHP 4: Include Oracle-oci7 support. Default DIR is ORACLE_HOME.

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/>)

-with-pgsql[=DIR]

PHP 3: Include PostgresSQL support. DIR is the PostgresSQL base install directory, which defaults to `/usr/local/pgsql`.

PHP 4: Include PostgreSQL support. DIR is the PostgreSQL base install directory, which defaults to `/usr/local/pgsql`. Set DIR to shared to build as a dl, or shared,DIR to build as a dl and still specify DIR.

See also [Postgres Configuration Directives](#) in the [configuration file](#).

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

-with-solid[=DIR]

PHP 3, PHP 4: Include Solid support. DIR is the Solid base install directory, defaults to /usr/local/solid
 Solid home page (<http://www.solidtech.com/>)

-with-sybase-ct[=DIR]

PHP 3, PHP 4: Include Sybase-CT support. DIR is the Sybase home directory, defaults to /home/sybase.
 See also [Sybase-CT Configuration](#) Directives in the [configuration file](#).

-with-sybase[=DIR]

PHP 3, PHP 4: Include Sybase-DB support. DIR is the Sybase home directory, which defaults to /home/sybase.
 See also [Sybase Configuration](#) Directives in the [configuration file](#).
 Sybase home page (<http://www.sybase.com/>)

-with-openlink[=DIR]

PHP 3, PHP 4: Include OpenLink ODBC support. DIR is the OpenLink base install directory, defaults to /usr/local/openlink.
 OpenLink Software's home page (<http://www.openlinksw.com/>)

-with-iodbc[=DIR]

PHP 3, PHP 4: Include iODBC support. DIR is the iODBC base install directory, defaults to /usr/local.
 This feature was first developed for iODBC Driver Manager, a freely redistributable ODBC driver manager which runs under many flavors of UNIX.
 FreeODBC home page (<http://users.ids.net/~bjepson/freeODBC/>) or iODBC home page (<http://www.iodbc.org/>)

-with-custom-odbc[=DIR]

PHP 3, PHP 4: 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=-lunix CUSTOM_ODBC_LIBS="-lplib -lodbc" ./configure -with-custom-odbc=/usr/lib/sqlany50

-disable-unified-odbc

PHP 3: Disable unified ODBC support. Only applicable if iODBC, Adabas, Solid, Velocis or a custom ODBC interface is enabled.

PHP 4: Option not available in PHP 4

The Unified ODBC module, which is a common interface to all the databases with ODBC-based interfaces, such as Solid, IBM DB2 and Adabas D. It also works for normal ODBC libraries. Has been tested with iODBC, Solid, Adabas D, IBM DB2 and Sybase SQL Anywhere. Requires that one (and only one) of these extensions or the Velocis extension 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-ibm-db2](#), [-with-adabas](#), [-with-velocis](#), or [-with-custom-odbc](#).

See also [Unified ODBC Configuration](#) Directives in the [configuration file](#).

-with-unixODBC[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include unixODBC support. DIR is the unixODBC base install directory, defaults to /usr/local.

-with-velocis[=DIR]

PHP 3, PHP 4: Include Velocis support. DIR is the Velocis base install directory, defaults to /usr/local/velocis.

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

Ecommerce

-with-ccvs[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Compile CCVS support into PHP 4. Please specify your CCVS base install directory as DIR.

-with-mck[=DIR]

PHP 3: Include Cybercash MCK support. DIR is the cybercash mck build directory, which defaults to /usr/src/mck-3.2.0.3-linux. For help, look in extra/cyberlib.

PHP 4: Option not available; use [-with-cybercash](#) instead.

-with-cybercash[=DIR]

PHP 3: Option not available; use [-with-mck](#) instead.

PHP 4: Include CyberCash support. DIR is the CyberCash MCK install directory.

-with-pfpro[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Verisign Payflow Pro support

Graphics

-enable-freetype-4bit-antialias-hack

PHP 3: Option not available in PHP 3

PHP 4: Include support for FreeType2 (experimental).

-with-gd[=DIR]

PHP 3: Include GD support (DIR is GD's install dir).

PHP 4: Include GD support (DIR is GD's install dir). Set DIR to shared to build as a dl, or shared,DIR to build as a dl and still specify DIR.

-without-gd

PHP 3, PHP 4: Disable GD support.

-with-imagick[=DIR]

PHP 3: Include ImageMagick support. DIR is the install directory, and if left out, PHP will try to find it on its own.
[experimental]

PHP 4: Option not available in PHP 4

-with-jpeg-dir[=DIR]

PHP 3: jpeg dir for pdflib 2.0

PHP 4: jpeg dir for pdflib 3.x

-with-png-dir[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: png dir for pdflib 3.x

-enable-t1lib

PHP 3: Enable t1lib support.

PHP 4: Option not available; use [*-with-t1lib*](#) instead.

-with-t1lib[=DIR]

PHP 3: Option not available; use [*-enable-t1lib*](#) instead.

PHP 4: Include T1lib support.

-with-tiff-dir[=DIR]

PHP 3: tiff dir for pdflib 2.0

PHP 4: tiff dir for pdflib 3.x

-with-ttf[=DIR]

PHP 3, PHP 4: Include FreeType support

-with-xpm-dir[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: xpm dir for gd-1.8+

Miscellaneous

These are being classified over time, where appropriate.

-disable-bcmath

PHP 3: Compile without BC arbitrary precision math functions. These functions allow you to operate with numbers outside of the ranges allowed by regular integers and floats; see [BCMath Arbitrary Precision Mathematics Functions](#) for more information.

PHP 4: Option not available; bcmath is not compiled in by default. Use [-enable-bcmath](#) to compile it in.

-disable-display-source

PHP 3: Compile without displaying source support

PHP 4: Option not available in PHP 4

-disable-libtool-lock

PHP 3: Option not available in PHP 3

PHP 4: avoid locking (might break parallel builds)

-disable-pear

PHP 3: Option not available in PHP 3

PHP 4: Do not install PEAR

-disable-pic

PHP 3: Option not available in PHP 3

PHP 4: Disable PIC for shared objects

-disable-posix

PHP 3: Option not available in PHP 3; use [-without-posix](#) instead.

PHP 4: Disable POSIX-like functions

-disable-rpath

PHP 3: Option not available in PHP 3

PHP 4: Disable passing additional runtime library search paths

-disable-session

PHP 3: Option not available in PHP 3

PHP 4: Disable session support

-enable-bcmath

PHP 3: Option not available in PHP 3; bcmath is compiled in by default. Use [-disable-bcmath](#) to disable it.

PHP 4: Compile with bc style precision math functions. Read README-BCMATH for instructions on how to get this module installed. These functions allow you to operate with numbers outside of the ranges allowed by regular integers and floats; see [BCMath Arbitrary Precision Mathematics Functions](#) for more information.

-enable-c9x-inline

PHP 3: Option not available in PHP 3

PHP 4: Enable C9x-inline semantics

-enable-calendar

PHP 3: Option not available in PHP 3

PHP 4: Enable support for calendar conversion

-enable-debug

PHP 3, PHP 4: Compile with debugging symbols.

-enable-debugger

PHP 3: Compile with remote debugging functions

PHP 4: Option not available in PHP 4

-enable-discard-path

PHP 3, PHP 4: If this is enabled, the PHP CGI binary can safely be placed outside of the web tree and people will not be able to circumvent .htaccess security.

-enable-dmalloc

PHP 3, PHP 4: Enable dmalloc

-enable-exif

PHP 3: Option not available in PHP 3

PHP 4: Enable exif support

-enable-experimental-zts

PHP 3: Option not available in PHP 3

PHP 4: This will most likely break your build

-enable-fast-install[=PKGS]

PHP 3: Option not available in PHP 3

PHP 4: optimize for fast installation [default=yes]

-enable-force-cgi-redirect

PHP 3, PHP 4: Enable the security check for internal server redirects. You should use this if you are running the CGI version with Apache.

-enable-inline-optimization

PHP 3: Option not available in PHP 3

PHP 4: If you have much memory and are using gcc, you might try this.

-enable-libgcc
 PHP 3: Option not available in PHP 3
 PHP 4: Enable explicitly linking against libgcc

-enable-maintainer-mode
 PHP 3, PHP 4: enable make rules and dependencies not useful (and sometimes confusing) to the casual installer

-enable-memory-limit
 PHP 3, PHP 4: Compile with memory limit support. [default=no]

-enable-safe-mode
 PHP 3, PHP 4: Enable safe mode by default.

-enable-satellite
 PHP 3: Option not available in PHP 3
 PHP 4: Enable CORBA support via Satellite (Requires ORBit)

-enable-shared[=PKGS]
 PHP 3: Option not available in PHP 3
 PHP 4: build shared libraries [default=yes]

-enable-sigchild
 PHP 3, PHP 4: Enable PHP's own SIGCHLD handler.

-enable-static[=PKGS]
 PHP 3: Option not available in PHP 3
 PHP 4: build static libraries [default=yes]

-enable-sysvsem
 PHP 3, PHP 4: Enable System V semaphore support.

-enable-sysvshm
 PHP 3, PHP 4: Enable the System V shared memory support

-enable-trans-sid
 PHP 3: Option not available in PHP 3
 PHP 4: Enable transparent session id propagation

-with-cdb[=DIR]
 PHP 3, PHP 4: Include CDB support

-with-config-file-path=PATH
 PHP 3: Sets the path in which to look for php3.ini. Defaults to /usr/local/lib
 PHP 4: Sets the path in which to look for php.ini. Defaults to /usr/local/lib

-with-cpdflib[=DIR]

PHP 3: Include ClibPDF support. DIR is the ClibPDF install directory, defaults to /usr/local.

PHP 4: Include cpdflib support (requires cpdflib >= 2). DIR is the cpdflib install directory, defaults to /usr.

-with-esooob[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Easysoft OOB support. DIR is the OOB base install directory, defaults to /usr/local/easysoft/oob/client.

-with-exec-dir[=DIR]

PHP 3, PHP 4: Only allow executables in DIR when in safe mode defaults to /usr/local/php/bin

-with-fdftk[=DIR]

PHP 3, PHP 4: Include fdftk support. DIR is the fdftk install directory, defaults to /usr/local.

-with-gnu-ld

PHP 3: Option not available in PHP 3

PHP 4: assume the C compiler uses GNU ld [default=no]

-with-icap[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include ICAP support.

-with-imap[=DIR]

PHP 3, PHP 4: Include IMAP support. DIR is the IMAP include and c-client.a directory.

-with-imsp[=DIR]

PHP 3: Include IMSP support (DIR is IMSP's include dir and libimsp.a dir).

PHP 4: Option not available in PHP 4

-with-java[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Java support. DIR is the base install directory for the JDK. This extension can only be built as a shared dl.

-with-kerberos[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include Kerberos support in IMAP.

-with-mcal[=DIR]

PHP 3, PHP 4: Include MCAL support.

-with-mcrypt[=DIR]

PHP 3, PHP 4: Include mcrypt support. DIR is the mcrypt install directory.

-with-mhash[=DIR]

PHP 3, PHP 4: Include mhash support. DIR is the mhash install directory.

-with-mm[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include mm support for session storage

-with-mod_charset

PHP 3, PHP 4: Enable transfer tables for mod_charset (Rus Apache).

-with-pdflib[=DIR]

PHP 3: Include pdflib support (tested with 0.6 and 2.0). DIR is the pdflib install directory, which defaults to /usr/local.

PHP 4: Include pdflib 3.x support. DIR is the pdflib install directory, which defaults to /usr/local.

-with-readline[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include readline support. DIR is the readline install directory.

-with-regex=TYPE

PHP 3: Option not available in PHP 3

PHP 4: regex library type: system, apache, php

-with-servlet[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include servlet support. DIR is the base install directory for the JSDK. This SAPI requires that the Java extension be built as a shared dl.

-with-swf[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include swf support

-with-system-regex

PHP 3: Do not use the bundled regex library

PHP 4: (deprecated) Use system regex library

-with-tsrn-pth[=pth-config]

PHP 3: Option not available in PHP 3

PHP 4: Use GNU Pth.

-with-tsrm-pthreads

PHP 3: Option not available in PHP 3
 PHP 4: Use POSIX threads (default)

-with-x

PHP 3: use the X Window System
 PHP 4: Option not available in PHP 4

-with-zlib-dir[=DIR]

PHP 3: zlib dir for pdflib 2.0 or include zlib support
 PHP 4: zlib dir for pdflib 3.x or include zlib support

-with-zlib[=DIR]

PHP 3, PHP 4: Include zlib support (requires zlib >= 1.0.9). DIR is the zlib install directory, defaults to /usr.

-without-pcre-regex

PHP 3: Don't include Perl Compatible Regular Expressions support
 PHP 4: Do not include Perl Compatible Regular Expressions support. Use ***--with-pcre-regex=DIR*** to specify DIR where PCRE's include and library files are located, if not using bundled library.

-without-posix

PHP 3: Don't include POSIX functions
 PHP 4: Option not available in PHP 4; use ***--disable-posix*** instead.

Networking

-with-curl[=DIR]

PHP 3: Option not available in PHP 3
 PHP 4: Include CURL support

-enable-ftp

PHP 3: Option not available; use ***--with-ftp*** instead.
 PHP 4: Enable FTP support

-with-ftp

PHP 3: Include FTP support.
 PHP 4: Option not available; use ***--enable-ftp*** instead

-disable-url-fopen-wrapper

PHP 3, PHP 4: Disable the URL-aware fopen wrapper that allows accessing files via http or ftp.

Warning

This switch is only available for PHP versions up to 4.0.3, newer versions provide an INI parameter called *allow_url_fopen* instead of forcing you to decide upon this feature at compile time.

-with-mod-dav=DIR

PHP 3, PHP 4: Include DAV support through Apache's mod_dav, DIR is mod_dav's installation directory (Apache module version only!)

-with-openssl[=DIR]

PHP 3, PHP 4: Include OpenSSL support in SNMP.

-with-snmp[=DIR]

PHP 3, PHP 4: Include SNMP support. DIR is the SNMP base install directory, defaults to searching through a number of common locations for the snmp install. Set DIR to shared to build as a dl, or shared,DIR to build as a dl and still specify DIR.

-enable-ucd-snmp-hack

PHP 3, PHP 4: Enable UCD SNMP hack

-enable-sockets

PHP 3: Option not available in PHP 3

PHP 4: Enable sockets support

-with-yaz[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include YAZ support (ANSI/NISO Z39.50). DIR is the YAZ bin install directory

-enable-yp

PHP 3: Option not available; use [-with-yp](#) instead.

PHP 4: Include YP support

-with-yp

PHP 3: Include YP support

PHP 4: Option not available; use [-enable-yp](#) instead.

PHP Behaviour

-enable-magic-quotes

PHP 3, PHP 4: Enable magic quotes by default.

-disable-short-tags

PHP 3, PHP 4: Disable the short-form <? start tag by default.

-enable-track-vars

PHP 3: Enable GET/POST/Cookie track variables by default.

PHP 4: Option not available in PHP 4; as of PHP 4.0.2, track_vars is always on.

Server

-with-aolserver-src=DIR

PHP 3: Option not available in PHP 3

PHP 4: Specify path to the source distribution of AOLserver

-with-aolserver=DIR

PHP 3: Option not available in PHP 3

PHP 4: Specify path to the installed AOLserver

-with-apache[=DIR]

PHP 3, PHP 4: Build Apache module. DIR is the top-level Apache build directory, defaults to /usr/local/etc/httpd.

-with-apxs[=FILE]

PHP 3, PHP 4: Build shared Apache module. FILE is the optional pathname to the Apache apxs tool; defaults to apxs.

-enable-versioning

PHP 3: Take advantage of versioning and scoping Provided by Solaris 2.x and Linux

PHP 4: Export only required symbols. See INSTALL for more information

-with-caudium[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Build PHP as a Pike module for use with the Caudium webserver. DIR is the Caudium base directory. If no directory is specified \$prefix/caudium/server is used. The prefix is controlled by the –prefix option and is /usr/local per default.

-with-fhttpd[=DIR]

PHP 3, PHP 4: Build fhttpd module. DIR is the fhttpd sources directory, defaults to /usr/local/src/fhttpd.

-with-nsapi=DIR

PHP 3: Option not available in PHP 3

PHP 4: Specify path to the installed Netscape

-with-phttpd=DIR

PHP 3: Option not available in PHP 3

PHP 4:

-with-pi3web=DIR

PHP 3: Option not available in PHP 3

PHP 4: Build PHP as a module for use with Pi3Web.

-with-roxen=DIR

PHP 3: Option not available in PHP 3

PHP 4: Build PHP as a Pike module. DIR is the base Roxen directory, normally /usr/local/roxen/server.

-enable-roxen-zts

PHP 3: Option not available in PHP 3

PHP 4: Build the Roxen module using Zend Thread Safety.

-with-thttpd=SRCDIR

PHP 3: Option not available in PHP 3

PHP 4:

-with-zeus=DIR

PHP 3: Option not available in PHP 3

PHP 4: Build PHP as an ISAPI module for use with Zeus.

Text and language

-with-aspell[=DIR]

PHP 3, PHP 4: Include ASPELL support.

-with-gettext[=DIR]

PHP 3, PHP 4: Include GNU gettext support. DIR is the gettext install directory, defaults to /usr/local

-with-pspell[=DIR]

PHP 3: Option not available in PHP 3

PHP 4: Include PSPELL support.

-with-recode[=DIR]

PHP 3: Include GNU recode support.

PHP 4: Include recode support. DIR is the recode install directory.

XML

`-with-dom[=DIR]`

PHP 3: Option not available in PHP 3

PHP 4: Include DOM support (requires libxml >= 2.0). DIR is the libxml install directory, defaults to /usr

`-enable-sablot-errors-descriptive`

PHP 3: Option not available in PHP 3

PHP 4: Enable Descriptive errors

`-with-sablot[=DIR]`

PHP 3: Option not available in PHP 3

PHP 4: Include Sablotron support

`-enable-wddx`

PHP 3: Option not available in PHP 3

PHP 4: Enable WDDX support

`-disable-xml`

PHP 3: Option not available in PHP 3; XML functionality is not built in by default. Use [-with-xml](#) to turn it on.

PHP 4: Disable XML support using bundled expat lib

`-with-xml`

PHP 3: Include XML support

PHP 4: Option not available; XML support is built in by default. Use [-disable-xml](#) to turn it off.

Installation on Windows 9x/Me/NT/2000 systems

There are two main ways to install PHP for Windows: either [manually](#) or by using the [InstallShield](#) installer.

If you have Microsoft Visual Studio, you can also [build](#) PHP from the original source code.

Once you have PHP installed on your Windows system, you may also want to [load various extensions](#) for added functionality.

Windows InstallShield

The Windows PHP installer available from the downloads page at <http://www.php.net/>, this installs the CGI version of PHP and, for IIS, PWS, and Xitami, configures the web server as well.

Install your selected HTTP server on your system and make sure that it works.

Run the executable installer and follow the instructions provided by the installation wizard. Two types of installation are supported - standard, which provides sensible defaults for all the settings it can, and advanced, which asks questions as it goes along.

The installation wizard gathers enough information to set up the `php.ini` file and configure the web server to use PHP. For IIS and also PWS on NT Workstation, a list of all the nodes on the server with script map settings is displayed, and you can choose those nodes to which you wish to add the PHP script mappings.

Once the installation has completed the installer will inform you if you need to restart your system, restart the server, or just start using PHP.

General Installation Steps

This install guide will help you manually install and configure PHP on your Windows 9x/Me/NT/2000 webservers. This guide was compiled by Bob Silva (mailto:bob_silva@mail.umesd.k12.or.us). The original version can be found at <http://www.umesd.k12.or.us/php/win32install.html>.

This guide provides manual installation support for:

- Personal Web Server 3 and 4 or newer
- Internet Information Server 3 and 4 or newer
- Apache 1.3.x
- OmniHTTPD 2.0b1 and up
- Oreilly Website Pro
- Xitami

PHP 4 for Windows comes in two flavours - a CGI executable (`php.exe`), and several SAPI modules (for example `php4isapi.dll`). The latter form is new to PHP 4, and provides significantly improved performance and some new functionality. However, please note that the SAPI modules are *NOT* yet considered to be production quality. The reason for this is that the PHP SAPI modules are using the thread-safe version of the PHP code, which is new to PHP 4, and has not yet been tested and polished enough to be considered completely stable, and there are actually a few known bugs. On the other hand, some people have reported very good results with the SAPI modules, even though we're not aware of anyone actually running it on a production site. In short - your mileage may vary; If you need absolute stability, trade the performance of the SAPI modules with the stability of the CGI executable.

If you choose one of the SAPI modules and use Windows 95, be sure to download the DCOM update from the Microsoft DCOM pages (<http://download.microsoft.com/msdownload/dcom/95/x86/en/dcom95.exe>). For the ISAPI module, an ISAPI 4.0 compliant Web server is required (tested on IIS 4.0, PWS 4.0 and IIS 5.0). IIS 3.0 is *NOT* supported; You should download and install the Windows NT 4.0 Option Pack with IIS 4.0 if you want native PHP support.

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:\PHP\" is a good start.
- Copy the file, 'php.ini-dist' to your '%WINDOWS%' directory on Windows 95/98 or to your '%SYSTEMROOT%' directory under Windows NT or Windows 2000 and rename it to 'php.ini'. Your '%WINDOWS%' or '%SYSTEMROOT%' directory is typically:
 c:\windows for Windows 95/98
 c:\winnt or c:\winnt40 for NT/2000 servers
- Edit your 'php.ini' file:
 - You will need to change the 'extension_dir' setting to point to your php-install-dir, or where you have placed your 'php_* .dll' files. ex: c:\php
 - If you are using OmniHTTPD, do not follow the next step. Set the 'doc_root' to point to your webservers document_root. ex: c:\apache\htdocs or c:\webroot
 - Choose which extensions you would like to load when PHP starts. You can uncomment the: 'extension=php_* .dll' lines in `php.ini` to load these extensions. Some extensions require you to have additional libraries installed on your

system for the module to work correctly. The PHP FAQ (<http://www.php.net/FAQ.php>) has more information on where to get supporting libraries. You can also load a module dynamically in your script using `dl()`. See the section about [Windows extensions](#).

- On PWS and IIS, you can set the `browscap.ini` to point to: '`c:\windows\system\inetsrv\browscap.ini`' on Windows 9x/Me and '`c:\winnt\system32\inetsrv\browscap.ini`' on NT/2000 Server. Additional information on using the browscap functionality in PHP can be found at this mirror (<http://php.netvision.net.il/browser-id.php3>), select the "source" button to see it in action.

Building from source

Before getting started, it is worthwhile answering the question: "Why is building on Windows so hard?" Two reasons come to mind:

1. Windows does not (yet) enjoy a large community of developers who are willing to freely share their source. As a direct result, the necessary investment in infrastructure required to support such development hasn't been made. By and large, what is available has been made possible by the porting of necessary utilities from Unix. Don't be surprised if some of this heritage shows through from time to time.
2. Pretty much all of the instructions that follow are of the "set and forget" variety. So sit back and try follow the instructions below as faithfully as you can.

Preparations

Before you get started, you have a lot to download....

- For starters, get the Cygwin toolkit from the closest cygwin (<http://sources.redhat.com/cygwin/download.html>) mirror site. This will provide you most of the popular GNU utilities used by the build process.
- Download the rest of the build tools you will need from the PHP site at <http://www.php.net/extra/win32build.zip>.
- Get the source code for the DNS name resolver used by PHP at http://www.php.net/extra/bindlib_w32.zip. This is a replacement for the `resolv.lib` library included in `win32build.zip`.
- If you don't already have an `unzip` utility, you will need one. A free version is available from InfoZip (<http://www.cdrom.com/pub/infozip/UnZip.html>).

Finally, you are going to need the source to PHP 4 itself. You can get the latest development version using anonymous CVS (<http://www.php.net/anoncvs.php>). If you get a snapshot (<http://snaps.php.net/>) or a source (<http://www.php.net/downloads.php>) tarball, you not only will have to untar and un gzip it, but you will have to convert the bare linefeeds to crlf's in the `*.dsp` and `*.dsw` files before Microsoft Visual C++ will have anything to do with them.

Note: Place the `Zend` and `TSRM` directories inside the `php4` directory in order for the projects to be found during the build process.

Putting it all together

- Follow the instructions for installing the `unzip` utility of your choosing.
- Execute `setup.exe` and follow the installation instructions. If you choose to install to a path other than `c:\cygnus`, let the build process know by setting the Cygwin environment variable. On Windows 95/98 setting an environment variable can be done by placing a line in your `autoexec.bat`. On Windows NT, go to My Computer => Control Panel => System and select the environment tab.

Warning

Make a temporary directory for Cygwin to use, otherwise many commands (particularly bison) will fail. On Windows 95/98, `mkdir C:\TMP`. For Windows NT, `mkdir %SystemDrive%\tmp`.

- Make a directory and unzip `win32build.zip` into it.
- Launch Microsoft Visual C++, and from the menu select Tools => Options. In the dialog, select the directories tab. Sequentially change the dropdown to Executables, Includes, and Library files, and ensure that `cygwin\bin`, `win32build\include`, and `win32build\lib` are in each list, respectively. (To add an entry, select a blank line at the end of the list and begin typing). Typical entries will look like this:
 - `c:\cygnus\bin`
 - `c:\php-win32build\include`
 - `c:\php-win32build\lib`

Press OK, and exit out of Visual C++.

- Make another directory and unzip `bindlib_w32.zip` into it. Decide whether you want to have debug symbols available (bindlib - Win32 Debug) or not (bindlib - Win32 Release). Build the appropriate configuration:
 - For GUI users, launch VC++, and then select File => Open Workspace and select bindlib. Then select Build=>Set Active Configuration and select the desired configuration. Finally select Build=>Rebuild All.
 - For command line users, make sure that you either have the C++ environment variables registered, or have run `vvars.bat`, and then execute one of the following:
 - `msdev bindlib.dsp /MAKE "bindlib - Win32 Debug"`
 - `msdev bindlib.dsp /MAKE "bindlib - Win32 Release"`
- At this point, you should have a usable `resolv.lib` in either your `Debug` or `Release` subdirectories. Copy this file into your `win32build\lib` directory over the file by the same name found in there.

Compiling

The best way to get started is to build the standalone/CGI version.

- For GUI users, launch VC++, and then select File => Open Workspace and select `php4ts`. Then select Build=>Set Active Configuration and select the desired configuration. Finally select Build=>Rebuild All.
- For command line users, make sure that you either have the C++ environment variables registered, or have run `vvars.bat`, and then execute one of the following:
 - `msdev php4ts.dsp /MAKE "php4ts - Win32 Debug_TS"`
 - `msdev php4ts.dsp /MAKE "php4ts - Win32 Release_TS"`
- At this point, you should have a usable `php.exe` in either your `Debug_TS` or `Release_TS` subdirectories.

Repeat the above steps with `php4isapi.dsp` (which can be found in `sapi\isapi`) in order to build the code necessary for integrating PHP with Microsoft IIS.

Installation of Windows extensions

After installing PHP and a webserver on Windows, you will probably want to install some extensions for added functionality. The following table describes some of the extensions available. As described in the manual installation steps, you can choose which extensions you would like to load when PHP starts by uncommenting the: 'extension=php_*.dll' lines in `php.ini`. Some extensions require you to have additional libraries installed on your system for the module to work correctly. The PHP FAQ (<http://www.php.net/FAQ.php>) has more information on where to get supporting libraries. You can also load a module dynamically in your script using `dl()`.

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

Note: In PHP 4.0.4pl1 MySQL, ODBC, FTP, Calendar, BCMath, COM, PCRE, Session, WDDX and XML support is *built-in*. You don't need to load any additional extensions in order to use these functions. See your distributions `README.txt` or `install.txt` for a list of built in modules.

Table 2-1. PHP Extensions

<code>php_calendar.dll</code>	Calendar conversion functions
<code>php_crypt.dll</code>	Crypt functions
<code>php_dbase.dll</code>	dBase functions
<code>php_dbm.dll</code>	Berkeley DB2 library
<code>php_filepro.dll</code>	Read-only access to Filepro databases
<code>php_gd.dll</code>	GD library functions for GIF manipulation
<code>php_hyperwave.dll</code>	HyperWave functions
<code>php_imap4r2.dll</code>	IMAP 4 functions
<code>php_ldap.dll</code>	LDAP functions
<code>php_msq11.dll</code>	mSQL 1 client
<code>php_msq12.dll</code>	mSQL 2 client
<code>php_mssql.dll</code>	MSSQL client (requires MSSQL DB-Libraries)
<code>php3_mysql.dll</code> (built into PHP 4)	MySQL functions
<code>php_nsmail.dll</code>	Netscape mail functions
<code>php_oc173.dll</code>	Oracle functions
<code>php_snmp.dll</code>	SNMP get and walk functions (NT only!)
<code>php_zlib.dll</code>	ZLib compression functions

Apache Server installs

This section contains notes and hints specific to Apache installs of PHP, both for [Unix](#) and [Windows](#) versions.

Details of installing PHP with Apache on Unix.

You can select arguments to add to the `configure` on line 8 below from the [Complete list of configure options](#).

Example 2-4. Installation Instructions (Apache Module Version)

1. `gunzip apache_1.3.x.tar.gz`
2. `tar xvf apache_1.3.x.tar`

```

3. gunzip php-x.x.x.tar.gz
4. tar xvf php-x.x.x.tar
5. cd apache_1.3.x
6. ./configure --prefix=/www
7. cd ../php-x.x.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. for PHP 3: ./configure --activate-module=src/modules/php3/libphp3.a
   for PHP 4: ./configure --activate-module=src/modules/php4/libphp4.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-x.x.x
16. for PHP 3: cp php3.ini-dist /usr/local/lib/php3.ini
   for PHP 4: cp php.ini-dist /usr/local/lib/php.ini

```

You can edit your .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:

```

For PHP 3: AddType application/x-httpd-php3 .php3
For PHP 4: AddType application/x-httpd-php .php

```

You can choose any extension you wish here. .php is simply the one we suggest. You can even include .html .

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.)

Different examples of compiling PHP for apache are as follows:

```
./configure --with-apxs --with-pgsql
```

This will create a libphp4.so shared library that is loaded into Apache using a LoadModule line in Apache's httpd.conf file. The PostgreSQL support is embedded into this libphp4.so library.

```
./configure --with-apxs --with-pgsql=shared
```

This will again create a libphp4.so shared library for Apache, but it will also create a pgsql.so shared library that is loaded into PHP either by using the extension directive in php.ini file or by loading it explicitly in a script using the dl() function.

```
./configure --with-apache=/path/to/apache_source --with-pgsql
```

This will create a `libmodphp4.a` library, a `mod_php4.c` and some accompanying files and copy this into the `src/modules/php4` directory in the Apache source tree. Then you compile Apache using `-activate-module=src/modules/php4/libphp4.a` and the Apache build system will create `libphp4.a` and link it statically into the `httpd` binary. The PostgreSQL support is included directly into this `httpd` binary, so the final result here is a single `httpd` binary that includes all of Apache and all of PHP.

```
./configure -with-apache=/path/to/apache_source -with-pgsql=shared
```

Same as before, except instead of including PostgreSQL support directly into the final `httpd` you will get a `pgsql.so` shared library that you can load into PHP from either the `php.ini` file or directly using `dl()`.

When choosing to build PHP in different ways, you should consider the advantages and drawbacks of each method. Building as a shared object will mean that you can compile apache separately, and don't have to recompile everything as you add to, or change, PHP. Building PHP into apache (static method) means that PHP will load and run faster. For more information, see the Apache webpage on DSO support (<http://www.apache.org/docs/dso.html>).

Details of installing PHP on Windows with Apache 1.3.x

There are two ways to set up PHP to work with Apache 1.3.x on Windows. One is to use the CGI binary (`php.exe`), the other is to use the Apache module `dll`. In either case you need to stop the Apache server, and edit your `srm.conf` or `httpd.conf` to configure Apache to work with PHP.

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

If you unzipped the PHP package to `C:\PHP\` as described in the [General Installation Steps](#) section, you need to insert these lines to your Apache conf file to set up the CGI binary:

- `ScriptAlias /php/ "c:/php/"`
- `AddType application/x-httdp-php .php .phtml`
- `Action application/x-httdp-php "/php/php.exe"`

If you would like to use PHP as a module in Apache, you should move `php4ts.dll` to the windows/system (for Windows 9x/Me) or winnt/system32 (for Windows NT/2000) directory, overwriting any older file. Then you should add the following two lines to your Apache conf file:

- `LoadModule php4_module c:/php/sapi/php4apache.dll`
- `AddType application/x-httdp-php .php .phtml`

To use the source code highlighting feature, simply create a PHP script file and stick this code in: `<?php show_source ("original_php_script.php"); ?>`. Substitute `original_php_script.php` with the name of the file you wish to show the source of. (This is the only way of doing so).

Note: On Win-Apache all backslashes in a path statement such as: "c:\directory\file.ext", must be converted to forward slashes.

CGI/Commandline Server Installs

The default is to build PHP as a CGI program. This creates a commandline interpreter, which can be used for CGI processing, or for non-web-related PHP scripting. 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.

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 3 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](#).

Note: **make bench** is only available for PHP 3.

fhttpd Server Installs

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.

Caudium Server installs

PHP 4 can be build as a Pike module for the Caudium webserver. Note that this is not supported with PHP 3. Follow the simple instructions below to install PHP 4 for Caudium.

Example 2-5. Caudium Installation Instructions

1. Make sure you have Caudium installed prior to attempting to install PHP 4. For PHP 4 to work correctly, you will need Pike 7.0.268 or newer. For the sake of this example we assume that Caudium is installed in **/opt/caudium/server/**.
2. Change directory to **php-x.y.z** (where x.y.z is the version number).
3. **./configure -with-caudium=/opt/caudium/server**
4. **make**
5. **make install**
6. Restart Caudium if it's currently running.
7. Log into the graphical configuration interface and go to the virtual server where you want to add PHP 4 support.
8. Click Add Module and locate and then add the PHP 4 Script Support module.
9. If the documentation says that the 'PHP 4 interpreter isn't available', make sure that you restarted the server. If you did check **/opt/caudium/logs/debug/default.1** for any errors related to **PHP4.so**. Also make sure that **caudium/server/lib/[pike-version]/PHP4.so** is present.
10. Configure the PHP Script Support module if needed.

You can of course compile your Caudium module with support for the various extensions available in PHP 4. See the [complete list of configure options](#) for an exhaustive rundown.

Note: When compiling PHP 4 with MySQL support you must make sure that the normal MySQL client code is used. Otherwise there might be conflicts if your Pike already has MySQL support. You do this by specifying a MySQL install directory [the --with-mysql option](#).

IIS/PWS Server installs

This section contains notes and hints specific to IIS (Microsoft Internet Information Server) installing PHP for [PWS/IIS 3](#), [PWS 4 or newer](#) and [IIS 4 or newer](#) versions.

Windows 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: `.php`
- Double click on the new string value and enter the path to `php.exe` in the value data field. ex: `c:\php\php.exe %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: `.php`
- 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:\php\php.exe -q %1`. (don't forget the `%1`).

- Exit Regedit.
- If using PWS on Windows, reboot to reload the registry.

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 and PWS 4 or newer

When installing PHP on Windows with PWS 4 or newer version, you have two options. One to set up the PHP CGI binary, the other is to use the ISAPI module dll.

If you choose the CGI binary, do the following:

- Edit the enclosed pws-php4cgi.reg file (look into the sapi dir) to reflect the location of your php.exe. Forward slashes should be escaped, for example:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="C:\\PHP\\php.exe"
```
- In the PWS Manager, right click on a given directory you want to add PHP support to, and select Properties. Check the 'Execute' checkbox, and confirm.

If you choose the ISAPI module, do the following:

- Edit the enclosed pws-php4isapi.reg file (look into the sapi dir) to reflect the location of your php4isapi.dll. Forward slashes should be escaped, for example:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="C:\\PHP\\sapi\\php4isapi.dll"
```
- In the PWS Manager, right click on a given directory you want to add PHP support to, and select Properties. Check the 'Execute' checkbox, and confirm.

Windows NT/2000 and IIS 4 or newer

To install PHP on an NT/2000 Server running IIS 4 or newer, follow these instructions. You have two options to set up PHP, using the CGI binary (php.exe) or with the ISAPI module.

In either case, you need to start the Microsoft Management Console (may appear as 'Internet Services Manager', either in your Windows NT 4.0 Option Pack branch or the Control Panel=>Administrative Tools under Windows 2000). Then right click on your Web server node (this will most probably appear as 'Default Web Server'), and select 'Properties'.

If you want to use the CGI binary, do the following:

- Under 'Home Directory', 'Virtual Directory', or 'Directory', click on the 'Configuration' button, and then enter the App Mappings tab.
- Click Add, and in the Executable box, type: c:\\php\\php.exe %s %s (assuming that you have unzipped PHP in c:\\php). 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. Leave 'Method exclusions' blank, and check the Script engine checkbox. You must repeat step 3 and 4 for each extension you want associated with PHP scripts. (.php 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.

To use the ISAPI module, do the following:

- If you don't want to perform HTTP Authentication using PHP, you can (and should) skip this step. Under ISAPI Filters, add a new ISAPI filter. Use PHP as the filter name, and supply a path to the php4isapi.dll.
- Under 'Home Directory', click on the 'Configuration' button. Add a new entry to the Application Mappings. Use the path to the php4isapi.dll as the Executable, supply .php as the extension, leave Method exclusions blank, and check the Script engine checkbox.
- Stop IIS completely
- Start IIS again

Netscape and iPlanet Enterprise Server Installs

To build PHP with NES or iPlanet web servers, enter the proper install directory for the `-with-nsapi = DIR` option. The default directory is usually `/opt/netscape/suitespot/`. Please also read `/php-xxx-version/sapi/nsapi/nsapi-readme.txt`.

Example 2-6. Installation Example for Netscape Enterprise on Solaris

Instructions for Sun Solaris 2.6 with Netscape Enterprise Server 3.6
From: bhager@invacare.com

1. Install the following packages from www.sunfreeware.com or another download site:

```
flex-2_5_4a-sol26-sparc-local
gcc-2_95_2-sol26-sparc-local
gzip-1.2.4-sol26-sparc-local
perl-5_005_03-sol26-sparc-local
bison-1_25-sol26-sparc-local
make-3_76_1-sol26-sparc-local
m4-1_4-sol26-sparc-local
autoconf-2.13
automake-1.4
mysql-3.23.24-beta (if you want mysql support)
tar-1.13 (GNU tar)
```

2. Make sure your path includes the proper directories
`PATH=.::/usr/local/bin:/usr/sbin:/usr/bin:/usr/ccs/bin`
`export PATH`
3. gunzip php-x.x.x.tar.gz (if you have a .gz dist, otherwise go to 4)
4. tar xvf php-x.x.x.tar
5. cd .../php-x.x.x
6. For the following step, make sure `/opt/netscape/suitespot/` is where your netscape server is installed. Otherwise, change to correct path:
`/configure -with-mysql=/usr/local/mysql -with-nsapi=/opt/netscape/suitespot/ -enable-track-vars -enable-libgcc`
7. make
8. make install

After performing the base install and reading the appropriate readme file, you may need to perform some additional configuration steps.

Firstly you may need to add some paths to the LD_LIBRARY_PATH environment for Netscape to find all the shared libs. This can best done in the start script for your Netscape server. Windows users can probably skip this step. The start script is often located in: /path/to/server/https-servername/start

You may also need to edit the configuration files that are located in:/path/to/server/https-servername/config/.

Example 2-7. Configuration Example for Netscape Enterprise

Configuration Instructions for Netscape Enterprise Server
From: bhager@invacare.com

1. Add the following line to mime.types:
type=magnus-internal/x-httdp-php exts=php

2. Add the following to obj.conf, shlib will vary depending on
your OS, for unix it will be something like
/opt/netscape/suitespot/bin/libphp4.so.

You should place the following lines after mime types init.
 Init fn="load-
 modules" funcs="php4_init,php4_close,php4_execute,php4_auth_trans" shlib="/php4/nsapiPHP4.dll"
 Init fn=php4_init errorString="Failed to initialize PHP!"

```
<object name="default">
.
.
.
#NOTE this next line should happen after all 'ObjectType' and before all 'AddLog' lines
Service fn="php4_execute" type="magnus-internal/x-httdp-php"
.
.
</Object>

<Object name="x-httdp-php">
ObjectType fn="force-type" type="magnus-internal/x-httdp-php"
Service fn=php4_execute
</Object>
```

Authentication configuration

PHP authentication cannot be used with any other authentication. ALL AUTHENTICATION IS PASSED TO YOUR PHP SCRIPT. To configure PHP Authentication for the entire server, add the following line:

```
<Object name="default">
AuthTrans fn=php4_auth_trans
.
.
.
</Object>
```

To use PHP Authentication on a single directory, add the following:

```
<Object ppath="d:\path\to\authenticated\dir\*">
AuthTrans fn=php4_auth_trans
</Object>
```

OmniHTTPd Server installs

This section contains notes and hints specific to OmniHTTPd.

OmniHTTPd 2.0b1 and up for Windows

This has got to be the easiest config there is:

- Step 1: Install OmniHTTPd 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 = .php | actual = c:\path-to-php-dir\php.exe, and use the Add button.
- Step 5: On the Mime tab, enter: virtual = wwwserver/stdcgi | actual = .php, and use the Add button.
- Step 6: Click OK

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

Note: Some OmniHTTPd packages come with built in PHP support. You can choose at setup time to do a custom setup, and uncheck the PHP component. We recommend you to use the latest PHP binaries. Some OmniHTTPd servers come with PHP 4 beta distributions, so you should choose not to set up the built in support, but install your own. If the server is already on your machine, use the Replace button in Step 4 and 5 to set the new, correct information.

Oreilly Website Pro Server installs

This section contains notes and hints specific to Oreilly Website Pro.

Oreilly Website Pro 2.5 and up for Windows

This list describes how to set up the PHP CGI binary or the ISAPI module to work with Oreilly Website Pro on Windows.

- Edit the Server Properties and select the tab "Mapping".
- From the List select "Associations" and enter the desired extension ("php") and the path to the CGI exe (ex. c:\php\php.exe) or the ISAPI dll file (ex. c:\php\sapi\php4isapi.dll).
- Select "Content Types" add the same extension ".php" and enter the content type. If you choose the CGI exe file, enter 'wwwserver/shellcgi', if you chose the ISAPI module, enter 'wwwserver/isapi' (both without quotes).

Xitami Server installs

This section contains notes and hints specific to Xitami.

Xitami for Windows

This list describes how to set up the PHP CGI binary to work with Xitami on Windows.

- Make sure the webserver is running, and point your browser to xitamis admin console (usually <http://127.0.0.1/admin>), and click on Configuration.
- Navigate to the Filters, and put the extension which php should parse (i.e. .php) into the field File extensions (.xxx).
- In Filter command or script put the path and name of your php executable i.e. c:\php\php.exe.
- Press the 'Save' icon.

Other web servers

PHP can be built to support a large number of web servers. Please see [Server-related options](#) for a full list of server-related configure options. The PHP CGI binaries are compatible with almost all webservers supporting the CGI interface.

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.php>

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.php>.

Other problems

If you are still stuck, someone on the PHP installation 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 installation mailing list, send an empty mail to php-install-subscribe@lists.php.net (<mailto:php-install-subscribe@lists.php.net>). The mailing list address is php-install@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

`allow_url_fopen boolean`

This option enables the URL-aware `fopen` wrappers that enable accessing URL object like files. Default wrappers are provided for the access of [remote files](#) using the `ftp` or `http` protocol, some extensions like `zlib` may register additional wrappers.

Note: This option was introduced immediately after the release of version 4.0.3. For versions up to and including 4.0.3 you can only disable this feature at compile time by using the configuration switch [`-disable-url-fopen-wrapper`](#).

`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-prepending.

`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 `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

On by default. If changed to Off scripts will be terminated as soon as they try to output something after a client has aborted their connection. **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. The default setting is 30.

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*register_globals* boolean

Tells whether or not to register the EGPCS (Environment, GET, POST, Cookie, Server) variables as global variables. You may want to turn this off if you don't want to clutter your scripts' global scope with user data. This makes the most sense when coupled with **track_vars** - in which case you can access all of the EGPCS variables through the `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS`, and `$HTTP_SERVER_VARS` arrays in the global scope.

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, then Environment, GET, POST, Cookie, and Server variables can be found in the global associative arrays `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS`, and `$HTTP_SERVER_VARS`.

Note that as of PHP 4.0.3, `track_vars` is always turned on.

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 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.default_host` string

The default server host to use when connecting to the database server if no other host is specified.

mysql.default_user string

The default user name to use when connecting to the database server if no other name is specified.

mysql.default_password string

The default password to use when connecting to the database server if no other password is specified.

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

msql.allow_persistent boolean

Whether to allow persistent mSQL connections.

msql.max_persistent integer

The maximum number of persistent mSQL connections per process.

msql.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.

SESAM Configuration Directives

sesam_oml string

Name of BS2000 PLAM library containing the loadable SESAM driver modules. Required for using SESAM functions. The BS2000 PLAM library must be set ACCESS=READ,SHARE=YES because it must be readable by the apache server's user id.

sesam_configfile string

Name of SESAM application configuration file. Required for using SESAM functions. The BS2000 file must be readable by the apache server's user id.

The application configuration file will usually contain a configuration like (see SESAM reference manual):

```
CNF=B
NAM=K
NOTYPE
```

sesam_messagecatalog string

Name of SESAM message catalog file. In most cases, this directive is not necessary. Only if the SESAM message file is not installed in the system's BS2000 message file table, it can be set with this directive.

The message catalog must be set ACCESS=READ,SHARE=YES because it must be readable by the apache server's user id.

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, and proper coding practices, it can give 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.

The configuration flexibility of PHP is equally rivalled by the code flexibility. PHP can be used to build complete server applications, with all the power of a shell user, or it can be used for simple server-side includes with little risk in a tightly controlled environment. How you build that environment, and how secure it is, is largely up to the PHP developer.

This chapter starts by explaining the different configuration option combinations and the situations they can be safely used. It then describes different considerations in coding for different levels of security, and ends with some general security advice.

Installed as 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) 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.php` 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.php`. Unfortunately, if the request is originally given in this form, no access checks are made by web server for file /secret/script.php, 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

`--enable-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.php` nor by redirection `http://my.host/dir/script.php`.

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/secretdir/script.php`. 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 php-script /cgi-bin/php
AddHandler php-script .php
```

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.php` does not result in opening a file under users home directory, but a file called `~user/doc.php` 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.php` will open a file called `doc.php` 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.php`.

`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.

Installed as an Apache module

When PHP is used as an Apache module it inherits Apache's user permissions (typically those of the "nobody" user). This has several impacts on security and authorization. For example, if you are using PHP to access a database, unless that database has built-in access control, you will have to make the database accessible to the "nobody" user. This means a malicious script could access and modify the database, even without a username and password. It's entirely possible that a web spider could stumble across a database administrator's web page, and drop all of your databases. You can protect against this with Apache authorization, or you can design your own access model using LDAP, .htaccess files, etc. and include that code as part of your PHP scripts.

Often, once security is established to the point where the PHP user (in this case, the apache user) has very little risk, it is discovered that PHP now has been prevented from writing virus files to user directories. Or perhaps it has been prevented from accessing or changing a non-public database. It has equally been secured from writing files that it should, or entering database transactions.

A frequent security mistake made at this point is to allow apache root permissions.

Escalating the Apache user's permissions to root is extremely dangerous and may compromise the entire system, so sudo'ing, chroot'ing ,or otherwise running as root should not be considered by those who are not security professionals.

Filesystem Security

PHP is subject to the security built into most server systems with respect to permissions on a file and directory basis. This allows you to control which files in the filesystem may be read. Care should be taken with any files which are world readable to ensure that they are safe for reading by all users who have access to that filesystem.

Since PHP was designed to allow user level access to the filesystem, it's entirely possible to write a PHP script that will allow you to read system files such as /etc/password, modify your ethernet connections, send massive printer jobs out, etc. This has some obvious implications, in that you need to ensure that the files that you read from and write to are the appropriate ones.

Consider the following script, where a user indicates that they'd like to delete a file in their home directory. This assumes a situation where a PHP web interface is regularly used for file management, so the Apache user is allowed to delete files in the user home directories.

Example 4-1. Poor variable checking leads to....

```
<?php
// remove a file from the user's home directory
$username = $user_submitted_name;
$homendir = "/home/$username";
$file_to_delete = "$userfile";
unlink ($homendir/$userfile);
echo "$file_to_delete has been deleted!";
?>
```

Since the username is postable from a user form, they can submit a username and file belonging to someone else, and delete files. In this case, you'd want to use some other form of authentication. Consider what could happen if the variables submitted were "../etc/" and "passwd". The code would then effectively read:

Example 4-2. ... A filesystem attack

```
<?php
// removes a file from anywhere on the hard drive that
// the PHP user has access to. If PHP has root access:
$username = "../etc/";
$homedir = "/home/../etc/";
$file_to_delete = "passwd";
unlink ("/home/../etc/passwd");
echo "/home/../etc/passwd has been deleted!";
?>
```

There are two important measures you should take to prevent these issues.

- Only allow limited permissions to the PHP web user binary.
- Check all variables which are submitted.

Here is an improved script:

Example 4-3. More secure file name checking

```
<?php
// removes a file from the hard drive that
// the PHP user has access to.
$username = $HTTP_REMOTE_USER; // use an authentication mechanism

$homedir = "/home/$username";

$file_to_delete = basename("$userfile"); // strip paths
unlink ($homedir/$file_to_delete);

$fp = fopen("/home/logging/filedelete.log", "+a"); //log the deletion
$logstring = "$HTTP_REMOTE_USER $homedir $file_to_delete";
fputs ($fp, $logstring);
fclose($fp);

echo "$file_to_delete has been deleted!";
?>
```

Alternately, you may prefer to write a more customized check:

Example 4-4. More secure file name checking

```
<?php
$username = getenv("REMOTE_USER");
$homedir = "/home/$username";

if (!ereg('^[^/][^/]*$', $userfile))
    die('bad filename'); //die, do not process

//etc...
?>
```

Depending on your operating system, there are a wide variety of files which you should be concerned about, including device entries (/dev/ or COM1), configuration files (/etc/ files and the .ini files), well known file storage areas (/home/, My Documents), etc. For this reason, it's usually easier to create a policy where you forbid everything except for what you explicitly allow.

Error Reporting

A standard attack tactic involves profiling a system by feeding it improper data, and checking for the kinds, and contexts, of the errors which are returned. This allows the system cracker to probe for information about the server, to determine possible weaknesses.

The PHP errors which are normally returned can be quite helpful to a developer who is trying to debug a script, indicating such things as the function or file that failed, the PHP file it failed in, and the line number which the failure occurred in. This is all information that can be exploited. It is not uncommon for a php developer to use `show_source()`, `highlight_string()`, or `highlight_file()` as a debugging measure, but in a live site, this can expose hidden variables, unchecked syntax, and other dangerous information.

For example, the very style of a generic error indicates a system is running PHP. If the attacker was looking at an .html page, and wanted to probe for the back-end (to look for known weaknesses in the system), by feeding it the wrong data they may be able to determine that a system was built with PHP.

A function error can indicate whether a system may be running a specific database engine, or give clues as to how a web page or programmed or designed. This allows for deeper investigation into open database ports, or to look for specific bugs or weaknesses in a web page. By feeding different pieces of bad data, for example, an attacker can determine the order of authentication in a script, (from the line number errors) as well as probe for exploits that may be exploited in different locations in the script.

A filesystem or general PHP error can indicate what permissions the webserver has, as well as the structure and organization of files on the web server. Developer written error code can aggravate this problem, leading to easy exploitation of formerly "hidden" information.

There are three major solutions to this issue. The first is to scrutinize all functions, and attempt to compensate for the bulk of the errors. The second is to disable error reporting entirely on the running code. The third is to use PHP's custom error handling functions to create your own error handler. Depending on your security policy, you may find all three to be applicable to your situation.

User Submitted Data

The greatest weakness in many PHP programs is not inherent in the language itself, but merely an issue of code not being written with security in mind. For this reason, you should always take the time to consider the implications of a given piece of code, to ascertain the possible damage if an unexpected variable is submitted to it.

Example 4-5. Dangerous Variable Usage

```
<?php
// remove a file from the user's home directory... or maybe
// somebody else's?
unlink ($evil_var);

// Write logging of their access... or maybe not?
fputs ($fp, $evil_var);

// Execute something trivial.. or rm -rf *?
system ($evil_var);
exec ($evil_var);

?>
```

You should always carefully examine your code to make sure that any variables being submitted from a web browser are being properly checked, and ask yourself the following questions:

- Will this script only affect the intended files?
- Can unusual or undesirable data be acted upon?

- Can this script be used in unintended ways?
- Can this be used in conjunction with other scripts in a negative manner?
- Will any transactions be adequately logged?

By adequately asking these questions while writing the script, rather than later, you prevent an unfortunate re-write when you need to increase your security. By starting out with this mindset, you won't guarantee the security of your system, but you can help improve it.

You may also want to consider turning off register_globals, magic_quotes, or other convenience settings which may confuse you as to the validity, source, or value of a given variable. Working with PHP in error_reporting(E_ALL) mode can also help warn you about variables being used before they are checked or initialized (so you can prevent unusual data from being operated upon).

General considerations

A completely secure system is a virtual impossibility, so an approach often used in the security profession is one of balancing risk and usability. If every variable submitted by a user required two forms of biometric validation (such as a retinal scan and a fingerprint), you would have an extremely high level of accountability. It would also take half an hour to fill out a fairly complex form, which would tend to encourage users to find ways of bypassing the security.

The best security is often inobtrusive enough to suit the requirements without the user being prevented from accomplishing their work, or over-burdening the code author with excessive complexity. Indeed, some security attacks are merely exploits of this kind of overly built security, which tends to erode over time.

A phrase worth remembering: A system is only as good as the weakest link in a chain. If all transactions are heavily logged based on time, location, transaction type, etc. but the user is only verified based on a single cookie, the validity of tying the users to the transaction log is severely weakened.

When testing, keep in mind that you will not be able to test all possibilities for even the simplest of pages. The input you may expect will be completely unrelated to the input given by a disgruntled employee, a cracker with months of time on their hands, or a housecat walking across the keyboard. This is why it's best to look at the code from a logical perspective, to discern where unexpected data can be introduced, and then follow how it is modified, reduced, or amplified.

The Internet is filled with people trying to make a name for themselves by breaking your code, crashing your site, posting inappropriate content, and otherwise making your day interesting. It doesn't matter if you have a small or large site, you are a target by simply being online, by having a server that can be connected to. Many cracking programs do not discern by size, they simply trawl massive IP blocks looking for victims. Try not to become one.

Keeping Current

PHP, like any other large system, is under constant scrutiny and improvement. Each new version will often include both major and minor changes to enhance and repair security flaws, configuration mishaps, and other issues that will affect the overall security and stability of your system.

Like other system-level scripting languages and programs, the best approach is to update often, and maintain awareness of the latest versions and their changes.

Part 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 XHTML or 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 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 second way is the generally preferred method, as it allows for the next generation of XHTML to be easily implemented with PHP.

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 <?php # 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:

- [array](#)
- [floating-point numbers](#)
- [integer](#)
- [object](#)
- [string](#)

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)
```

The size of an integer is platform-dependent, although a maximum value of about 2 billion is the usual value (that's 32 bits signed).

Floating point numbers

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

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

The size of a floating point number is platform-dependent, although a maximum of $\sim 1.8 \times 10^{308}$ with a precision of roughly 14 decimal digits is a common value (that's 64 bit IEEE format).

Warning

It is quite usual that simple decimal fractions like 0.1 or 0.7 cannot be converted into their internal binary counterparts without a little loss of precision. This can lead to confusing results: for example, `floor((0.1+0.7)*10)` will usually return 7 instead of the expected 8 as the result of the internal representation really being something like $7.9999999999\dots$

This is related to the fact that it is impossible to exactly express some fractions in decimal notation with a finite number of digits. For instance, $1/3$ in decimal form becomes $0.333333\dots$

So never trust floating number results to the last digit and never compare floating point numbers for equality. If you really need higher precision, you should use the [arbitrary precision math functions](#) or [gmp](#) functions instead.

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	linefeed (LF or 0x0A in ASCII)
\r	carriage return (CR or 0x0D in ASCII)
\t	horizontal tab (HT or 0x09 in ASCII)
\\"	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

If you attempt to escape any other character, both the backslash and the character will be output. In PHP 3, a warning will be issued at the `E_NOTICE` level when this happens. In PHP 4, no warning is generated.

The second way to delimit a string uses the single-quote ('') character. When a string is enclosed in single quotes, the only escapes that will be understood are "\\" and "\'". This is for convenience, so that you can have single-quotes and backslashes in a single-quoted string. Variables will *not* be expanded inside a single-quoted string.

Another way to delimit strings is by using here doc syntax ("<<"). One should provide an identifier after <<, then the string, and then the same identifier to close the quotation. The closing identifier *must* begin in the first column of the line. The label used must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.

Here doc text behaves just like a double-quoted string, without the double-quotes. This means that you do not need to escape quotes in your here docs, but you can still use the escape codes listed above. Variables are expanded, but the same care must be taken when expressing complex variables inside a here doc as with strings.

Example 6-1. Here doc string quoting example

```
<?php
$str = <<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;

/* More complex example, with variables. */
class foo {
    var $foo;
    var $bar;

    function foo() {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
```

```
$name = 'MyName';

echo <<EOT
My name is "$name". I am printing some $foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should print a capital 'A': \x41
EOT;
?>
```

Note: Here doc support was added in PHP 4.

Strings may be concatenated using the '.' (dot) operator. Note that the '+' (addition) operator will not work for this. Please see [String operators](#) for more information.

Characters within strings may be accessed by treating the string as a numerically-indexed array of characters, using C-like syntax. See below for examples.

Example 6-2. Some string examples

```
<?php
/* Assigning a string. */
$str = "This is a string";

/* Appending to it. */
$str = $str . " with some more text";

/* Another way to append, includes an escaped newline. */
$str .= " and a newline at the end.\n";

/* This string will end up being '<p>Number: 9</p>' */
$num = 9;
$str = "<p>Number: $num</p>";

/* This one will be '<p>Number: $num</p>' */
$num = 9;
$str = '<p>Number: $num</p>';

/* Get the first character of a string */
$str = 'This is a test.';
$first = $str[0];

/* Get the last character of a string. */
$str = 'This is still a test.';
$last = $str[strlen($str)-1];
?>
```

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)`.

If you would like to test any of the examples in this section, you can cut and paste the examples and insert the following line to see for yourself what's going on:

```
echo "\$foo==\$foo; type is " . gettype ($foo) . "<br>\n";
```

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. When you assign a value to an array variable using empty brackets, the value will be added onto the end of 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 associative indices)
$a[3]["bar"] = $f;                  # (you can mix numeric and associative indices)

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

In PHP 3 it is not possible to reference multidimensional arrays directly within strings. For instance, the following will not have the desired result:

```
$a[3]['bar'] = 'Bob';
echo "This won't work: $a[3][bar]";
```

In PHP 3, the above will output `This won't work: Array[bar]`. The string concatenation operator, however, can be used to overcome this:

```
$a[3]['bar'] = 'Bob';
echo "This will work: " . $a[3][bar];
```

In PHP 4, however, the whole problem may be circumvented by enclosing the array reference (inside the string) in curly braces:

```
$a[3]['bar'] = 'Bob';
echo "This will work: {$a[3][bar]}";
```

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:

```
<?php
$a = array(
    "apple"  => array(
        "color"  => "red",
        "taste"  => "sweet",
        "shape"  => "round"
    ),
    "orange"  => array(
        "color"  => "orange",
        "taste"  => "tart",
        "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.

```
<?php
class foo {
    function do_foo() {
        echo "Doing foo.";
    }
}

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

For a full discussion, please read the section [Classes and Objects](#).

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\(\)](#).

If you would like to test any of the examples in this section, you can cut and paste the examples and insert the following line to see for yourself what's going on:

```
echo "\$foo==\$foo; type is " . gettype ($foo) . "<br>\n";
```

Note: The behaviour of an automatic conversion to array is currently undefined.

```
$a = 1; // $a is an integer
$a[0] = "f"; // $a becomes an array, with $a[0] holding "f"
```

While the above example may seem like it should clearly result in \$a becoming an array, the first element of which is 'f', consider this:

```
$a = "1"; // $a is a string
$a[0] = "f"; // What about string offsets? What happens?
```

Since PHP supports indexing into strings via offsets using the same syntax as array indexing, the example above leads to a problem: should \$a become an array with its first element being "f", or should "f" become the first character of the string \$a?

For this reason, as of PHP 3.0.12 and PHP 4.0b3-RC4, the result of this automatic conversion is considered to be undefined. Fixes are, however, being discussed.

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.

When casting from a scalar or a string variable to an array, the variable will become the first element of the array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0]; // outputs 'ciao'
```

When casting from a scalar or a string variable to an object, the variable will become an attribute of the object; the attribute name will be 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar; // outputs 'ciao'
```

Chapter 7. Variables

Basics

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

Variable names follow the same rules as other labels in PHP. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus:

'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Note: For our purposes here, a letter is a-z, A-Z, and the ASCII characters from 127 through 255 (0x7f-0xff).

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // outputs "Bob, Joe"

$4site = 'not yet';    // invalid; starts with a number
$_4site = 'not yet';   // valid; starts with an underscore
$stäyte = 'mansikka'; // valid; 'ä' is ASCII 228.
```

In PHP 3, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other. For more information on this kind of assignment, see [Expressions](#).

PHP 4 offers another way to assign values to variables: *assign by reference*. This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable. Changes to the new variable affect the original, and vice versa. This also means that no copying is performed; thus, the assignment happens more quickly. However, any speedup will likely be noticed only in tight loops or when assigning large arrays or objects.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable). For instance, the following code snippet outputs 'My name is Bob' twice:

```
<?php
$foo = 'Bob';           // Assign the value 'Bob' to $foo
$bar = &$foo;          // Reference $foo via $bar.
$bar = "My name is $bar"; // Alter $bar...
echo $foo;             // $foo is altered too.
echo $bar;
?>
```

One important thing to note is that only named variables may be assigned by reference.

```
<?php
$foo = 25;
$bar = &$foo;          // This is a valid assignment.
$bar = &(24 * 7);     // Invalid; references an unnamed expression.

function test() {
    return 25;
}

$bar = &test();        // Invalid.
?>
```

Predefined variables

PHP provides a large number of predefined variables to any script which it runs. Many of these variables, however, cannot be fully documented as they are dependent upon which server is running, the version and setup of the server, and other factors. Some of these variables will not be available when PHP is run on the command-line.

Despite these factors, here is a list of predefined variables available under a stock installation of PHP 3 running as a module under a stock installation of Apache (<http://www.apache.org/>) 1.3.6.

For a list of all predefined variables (and lots of other useful information), please see (and use) **`phpinfo()`**.

Note: This list is neither exhaustive nor intended to be. It is simply a guideline as to what sorts of predefined variables you can expect to have access to in your script.

Apache variables

These variables are created by the Apache (<http://www.apache.org/>) webserver. If you are running another webserver, there is no guarantee that it will provide the same variables; it may omit some, or provide others not listed here. That said, a large number of these variables are accounted for in the CGI 1.1 specification (<http://hoohoo.ncsa.uiuc.edu/cgi/env.html>), so you should be able to expect those.

Note that few, if any, of these will be available (or indeed have any meaning) if running PHP on the command line.

GATEWAY_INTERFACE

What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.

SERVER_NAME

The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.

SERVER_SOFTWARE

Server identification string, given in the headers when responding to requests.

SERVER_PROTOCOL

Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';

REQUEST_METHOD

Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.

QUERY_STRING

The query string, if any, via which the page was accessed.

DOCUMENT_ROOT

The document root directory under which the current script is executing, as defined in the server's configuration file.

HTTP_ACCEPT

Contents of the `Accept`: header from the current request, if there is one.

HTTP_ACCEPT_CHARSET

Contents of the `Accept-Charset`: header from the current request, if there is one. Example: 'iso-8859-1,*;utf-8'.

HTTP_ENCODING

Contents of the `Accept-Encoding`: header from the current request, if there is one. Example: 'gzip'.

HTTP_ACCEPT_LANGUAGE

Contents of the `Accept-Language`: header from the current request, if there is one. Example: 'en'.

HTTP_CONNECTION

Contents of the `Connection`: header from the current request, if there is one. Example: 'Keep-Alive'.

HTTP_HOST

Contents of the `Host`: header from the current request, if there is one.

HTTP_REFERER

The address of the page (if any) which referred the browser to the current page. This is set by the user's browser; not all browsers will set this.

HTTP_USER_AGENT

Contents of the `User-Agent`: header from the current request, if there is one. This is a string denoting the browser software being used to view the current page; i.e. `Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)`. Among other things, you can use this value with `get_browser()` to tailor your page's functionality to the capabilities of the user's browser.

REMOTE_ADDR

The IP address from which the user is viewing the current page.

REMOTE_PORT

The port being used on the user's machine to communicate with the web server.

SCRIPT_FILENAME

The absolute pathname of the currently executing script.

SERVER_ADMIN

The value given to the `SERVER_ADMIN` (for Apache) directive in the web server configuration file. If the script is running on a virtual host, this will be the value defined for that virtual host.

SERVER_PORT

The port on the server machine being used by the web server for communication. For default setups, this will be '80'; using SSL, for instance, will change this to whatever your defined secure HTTP port is.

SERVER_SIGNATURE

String containing the server version and virtual host name which are added to server-generated pages, if enabled.

PATH_TRANSLATED

Filesystem- (not document root-) based path to the current script, after the server has done any virtual-to-real mapping.

SCRIPT_NAME

Contains the current script's path. This is useful for pages which need to point to themselves.

REQUEST_URI

The URI which was given in order to access this page; for instance, '/index.html'.

Environment variables

These variables are imported into PHP's global namespace from the environment under which the PHP parser is running. Many are provided by the shell under which PHP is running and different systems are likely running different kinds of shells, a definitive list is impossible. Please see your shell's documentation for a list of defined environment variables.

Other environment variables include the CGI variables, placed there regardless of whether PHP is running as a server module or CGI processor.

PHP variables

These variables are created by PHP itself. The `$HTTP_*_VARS` variables are available only if the [track_vars](#) configuration is turned on. When enabled, the variables are always set, even if they are empty arrays. This prevents a malicious user from spoofing these variables.

Note: As of PHP 4.0.3, [track_vars](#) is always turned on, regardless of the configuration file setting.

If the [register_globals](#) directive is set, then these variables will also be made available in the global scope of the script; i.e., separate from the `$HTTP_*_VARS` arrays. This feature should be used with care, and turned off if possible; while the `$HTTP_*_VARS` variables are safe, the bare global equivalents can be overwritten by user input, with possibly malicious intent. If you cannot turn off [register_globals](#), you must take whatever steps are necessary to ensure that the data you are using is safe.

argv

Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.

argc

Contains the number of command line parameters passed to the script (if run on the command line).

PHP_SELF

The filename of the currently executing script, relative to the document root. If PHP is running as a command-line processor, this variable is not available.

HTTP_COOKIE_VARS

An associative array of variables passed to the current script via HTTP cookies.

HTTP_GET_VARS

An associative array of variables passed to the current script via the HTTP GET method.

HTTP_POST_VARS

An associative array of variables passed to the current script via the HTTP POST method.

HTTP_POST_FILES

An associative array of variables containing information about files uploaded via the HTTP POST method. See [POST method uploads](#) for information on the contents of `$HTTP_POST_FILES`.

`$HTTP_POST_FILES` is available only in PHP 4.0.0 and later.

HTTP_ENV_VARS

An associative array of variables passed to the current script via the parent environment.

HTTP_SERVER_VARS

An associative array of variables passed to the current script from the HTTP server. These variables are analogous to the Apache variables described above.

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. This single scope spans included and required files as well. For example:

```
$a = 1;
include "b.inc";
```

Here the \$a variable will be available within the included b.inc script. 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 also provide one way to deal with recursive functions. 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, using the static variable \$count to know when to stop:

```
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. i.e.

```
$$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";
```

i.e. 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. If the `track_vars` configuration option is turned on, then these variables will be located in the associative arrays `$HTTP_POST_VARS`, `$HTTP_GET_VARS`, and/or `$HTTP_POST_FILES`, according to the source of the variable in question.

For more information on these variables, please read [Predefined variables](#).

Example 7-1. Simple form variable

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

When the above form is submitted, the value from the text input will be available in `$HTTP_POST_VARS['username']`. If the `register_globals` configuration directive is turned on, then the variable will also be available as `$username` in the global scope.

PHP also understands arrays in the context of form variables. 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
        <option value="stuttgarter">Stuttgarter Schwabenbräu
    </select>
    <input type="submit">
</form>
```

In PHP 3, the array form variable usage is limited to single-dimensional arrays. In PHP 4, no such restriction applies.

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). 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.

Dots in incoming variable names

Typically, PHP does not alter the names of variables when they are passed into a script. However, it should be noted that the dot (period, full stop) is not a valid character in a PHP variable name. For the reason, look at it:

```
$varname.ext; /* invalid variable name */
```

Now, what the parser sees is a variable named `$varname`, followed by the string concatenation operator, followed by the barestring (i.e. unquoted string which doesn't match any known key or reserved words) `'ext'`. Obviously, this doesn't have the intended result.

For this reason, it is important to note that PHP will automatically replace any dots in incoming variable names with underscores.

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.

E_ALL

All of the E_* constants rolled into one. If used with **error_reporting()**, will cause any and all problems noticed by PHP to be reported.

The E_* constants are typically used with the **error_reporting()** function for setting the error reporting level. See all these constants at [Error handling](#).

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 = 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 it 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 */
$e = $d = ++$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
\$a + \$b	Addition	Sum of \$a and \$b.
\$a - \$b	Subtraction	Difference of \$a and \$b.
\$a * \$b	Multiplication	Product of \$a and \$b.
\$a / \$b	Division	Quotient of \$a and \$b.
\$a % \$b	Modulus	Remainder of \$a divided by \$b.

The division operator ("/") returns an integer value (the result of an integer division) if the two operands are integers (or strings that get converted to integers) and the quotient is an integer. If either operand is a floating-point value, or the operation results in a non-integer value, a floating-point value is returned.

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 left operand gets set to the value of the expression on the right (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!";
```

Note that the assignment copies the original variable to the new one (assignment by value), so changes to one will not affect the other. This may also have relevance if you need to copy something like a large array inside a tight loop. PHP 4 supports assignment by reference, using the `$var = &$othervar;` syntax, but this is not possible in PHP 3. 'Assignment by reference' means that both variables end up pointing at the same data, and nothing is copied anywhere. To learn more about references, please read [References explained](#).

Bitwise Operators

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

Table 10-2. Bitwise Operators

Example	Name	Result
\$a & \$b	And	Bits that are set in both \$a and \$b are set.

Example	Name	Result
<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 << \$b</code>	Shift left	Shift the bits of \$a \$b steps to the left (each step means "multiply by two")
<code>\$a >> \$b</code>	Shift right	Shift the bits of \$a \$b steps to the right (each step means "divide by two")

Comparison Operators

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

Table 10-3. Comparison Operators

Example	Name	Result
<code>\$a == \$b</code>	Equal	True if \$a is equal to \$b.
<code>\$a === \$b</code>	Identical	True if \$a is equal to \$b, and they are of the same type. (PHP 4 only)
<code>\$a != \$b</code>	Not equal	True if \$a is not equal to \$b.
<code>\$a !== \$b</code>	Not identical	True if \$a is not equal to \$b, or they are not of the same type. (PHP 4 only)
<code>\$a < \$b</code>	Less than	True if \$a is strictly less than \$b.
<code>\$a > \$b</code>	Greater than	True if \$a is strictly greater than \$b.
<code>\$a <= \$b</code>	Less than or equal to	True if \$a is less than or equal to \$b.
<code>\$a >= \$b</code>	Greater than or equal to	True if \$a is greater than or equal to \$b.

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

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

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

Error Control Operators

PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

If the `track_errors` feature is enabled, any error message generated by the expression will be saved in the global variable `$php_errormsg`. This variable will be overwritten on each error, so check early if you want to use it.

```
<?php
/* Intentional SQL error (extra quote): */
$res = @mysql_query ("select name, code from 'namelist') or
      die ("Query failed: error was '$php_errormsg'");
?>
```

See also [error_reporting\(\)](#).

Warning

Currently the "@" error-control operator prefix will even disable error reporting for critical errors that will terminate script execution. Among other things, this means that if you use "@" to suppress errors from a certain function and either it isn't available or has been mistyped, the script will die right there with no indication as to why.

Execution Operators

PHP supports one execution operator: backticks (`). Note that these are not single-quotes! PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable).

```
$output = `ls -al`;
echo "<pre>$output</pre>";
```

Note: The backtick operator is disabled when [safe mode](#) is enabled.

See also [system\(\)](#), [passthru\(\)](#), [exec\(\)](#), [popen\(\)](#), and [escapeshellcmd\(\)](#).

Incrementing/Decrementing Operators

PHP supports C-style pre- and post-increment and decrement operators.

Table 10-4. Increment/decrement Operators

Example	Name	Effect
<code>++\$a</code>	Pre-increment	Increments \$a by one, then returns \$a.
<code>\$a++</code>	Post-increment	Returns \$a, then increments \$a by one.
<code>-\$a</code>	Pre-decrement	Decrements \$a by one, then returns \$a.
<code>\$a-</code>	Post-decrement	Returns \$a, then decrements \$a by one.

Here's a simple example script:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be 5: " . $a++ . "<br>\n";
echo "Should be 6: " . $a . "<br>\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be 6: " . ++$a . "<br>\n";
echo "Should be 6: " . $a . "<br>\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be 5: " . $a- . "<br>\n";
echo "Should be 4: " . $a . "<br>\n";
```

```

echo "<h3>Predecrement</h3>" ;
$a = 5;
echo "Should be 4: " . -$a . "<br>\n";
echo "Should be 4: " . $a . "<br>\n";
?>

```

Logical Operators

Table 10-5. Logical Operators

Example	Name	Result
\$a and \$b	And	True if both \$a and \$b are true.
\$a or \$b	Or	True if either \$a or \$b is true.
\$a xor \$b	Xor	True if either \$a or \$b is true, but not both.
! \$a	Not	True if \$a is not true.
\$a && \$b	And	True if 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 [Operator Precedence](#).)

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. Parentheses may be used to force precedence, if necessary. For instance: `(1 + 5) * 3` evaluates to 18.

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

Table 10-6. Operator Precedence

Associativity	Operators
left	,
left	or
left	xor
left	and
right	print
left	= += -= *= /= .= %= &= = ^= ~= <= >=
left	? :
left	
left	&&
left	
left	^
left	&
non-associative	== != === !==
non-associative	< <= > >=

Associativity	Operators
left	<< >>
left	+ - .
left	* / %
right	! ~ ++ - (int) (double) (string) (array) (object) @
right	[
non-associative	new

String Operators

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side. Please read [Assignment Operators](#) for more information.

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

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


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 [elseif](#)).

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 control structures

PHP offers an alternative syntax for some of its control structures; namely, `if`, `while`, `for`, `foreach`, and `switch`. In each case, the basic form of the alternate syntax is to change the opening brace to a colon (`:`) and the closing brace to `endif;`, `endwhile;`, `endfor;`, `endforeach;`, or `endswitch;`, respectively.

```
<?php if ($a == 5): ?>
A is equal to 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;
```

See also [while](#), [for](#), and [if](#) for further examples.

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 3 has no such construct; PHP 4 does (see [foreach](#)). In PHP 3, you can combine [while](#) with the [list\(\)](#) and [each\(\)](#) functions to achieve the same effect. See the documentation for these functions for an example.

foreach

PHP 4 (not PHP 3) includes a `foreach` construct, much like perl and some other languages. This simply gives an easy way to iterate over arrays. There are two syntaxes; the second is a minor but useful extension of the first:

```
foreach(array_expression as $value) statement
foreach(array_expression as $key => $value) statement
```

The first form loops over the array given by `array_expression`. On each loop, the value of the current element is assigned to `$value` and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

The second form does the same thing, except that the current element's key will be assigned to the variable `$key` on each loop.

Note: When `foreach` first starts executing, the internal array pointer is automatically reset to the first element of the array. This means that you do not need to call [reset\(\)](#) before a `foreach` loop.

Note: Also note that `foreach` operates on a copy of the specified array, not the array itself, therefore the array pointer is not modified like with the `each` construct.

You may have noticed that the following are functionally identical:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}

foreach ($arr as $value) {
    echo "Value: $value<br>\n";
}
```

The following are also functionally identical:

```
reset ($arr);
while (list($key, $value) = each ($arr)) {
    echo "Key: $key; Value: $value<br>\n";
}

foreach ($arr as $key => $value) {
    echo "Key: $key; Value: $value<br>\n";
}
```

Some more examples to demonstrate usages:

```
/* foreach example 1: value only */

$a = array (1, 2, 3, 17);

foreach ($a as $v) {
    print "Current value of \$a: $v.\n";
}

/* foreach example 2: value (with key printed for illustration) */

$a = array (1, 2, 3, 17);

$i = 0; /* for illustrative purposes only */

foreach($a as $v) {
    print "\$a[$i] => $k.\n";
}

/* foreach example 3: key and value */

$a = array (
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach($a as $k => $v) {
    print "\$a[$k] => $v.\n";
}
```

break

`break` ends execution of the current `for`, `while`, or `switch` structure.

`break` accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (), $val) = each ($arr)) {
    if ($val == 'stop') {
        break;      /* You could also write 'break 1;' here. */
    }
    echo "$val<br>\n";
}

/* Using the optional argument. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1; /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2; /* Exit the switch and the while. */
    default:
        break;
    }
}
```

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.

`continue` accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

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

$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "    Middle<br>\n";
        while (1) {
            echo "        Inner<br>\n";
            continue 3;
        }
        echo "    This never gets output.<br>\n";
    }
    echo "Neither does this.<br>\n";
}
```

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).

In a `switch` statement, the condition is evaluated only once and the result is compared to each `case` statement. In an `elseif` statement, the condition is evaluated again. If your condition is more complicated than a simple compare and/or is in a tight loop, a `switch` may be faster.

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 simple type, that is, integer or floating-point numbers and strings. Arrays or objects cannot be used here unless they are dereferenced to a simple type.

The alternative syntax for control structures is supported with switches. For more information, see [Alternative syntax for control structures](#).

```
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";
endswitch;
```

require()

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

If "URL fopen wrappers" are enabled in PHP (which they are in the default configuration), you can specify the file to be **require()**ed using an URL instead of a local pathname. See [Remote files](#) and **fopen()** for more information.

An important note about how this works is that when a file is **include()**ed or **require()**ed, parsing drops out of PHP mode and into HTML mode at the beginning of the target file, and resumes PHP mode again at the end. For this reason, any code inside the target file which should be executed as PHP code must be enclosed within [valid PHP start and end tags](#).

require() is not actually a function in PHP; rather, it is a language construct. It is subject to some different rules than functions are. For instance, **require()** is not subject to any containing control structures. For another, it does not return any value; attempting to read a return value from a **require()** call results in a parse error.

Unlike **include()**, **require()** will *always* read in the target file, *even if the line it's on never executes*. If you want to conditionally include a file, use **include()**. The conditional statement won't affect the **require()**. However, if the line on which the **require()** occurs is not executed, neither will any of the code in the target file be executed.

Similarly, looping structures do not affect the behaviour of **require()**. Although the code contained in the target file is still subject to the loop, the **require()** itself happens only once.

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');
```

When a file is **require()**ed, the code it contains inherits the variable scope of the line on which the **require()** occurs. Any variables available at that line in the calling file will be available within the called file. If the **require()** occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function.

If the **require()**ed file is called via HTTP using the fopen wrappers, and if the target server interprets the target file as PHP code, variables may be passed to the **require()**ed file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as **require()**ing the file and having it inherit the parent file's variable scope; the script is actually being run on the remote server and the result is then being included into the local script.

```
/* This example assumes that someserver is configured to parse .php
 * files and not .txt files. Also, 'works' here means that the variables
 * $varone and $vartwo are available within the require()ed file. */

/* Won't work; file.txt wasn't handled by someserver. */
require ("http://someserver/file.txt?varone=1&vartwo=2");

/* Won't work; looks for a file named 'file.php?varone=1&vartwo=2'
 * on the local filesystem. */
require ("file.php?varone=1&vartwo=2");

/* Works. */
require ("http://someserver/file.php?varone=1&vartwo=2");

$varone = 1;
$vartwo = 2;
require ("file.txt"); /* Works. */
require ("file.php"); /* Works. */
```

In PHP 3, it is possible to execute a **return** statement inside a **require()**ed file, as long as that statement occurs in the global scope of the **require()**ed file. It may not occur within any block (meaning inside braces `{ }`). In PHP 4, however, this ability has been discontinued. If you need this functionality, see **include()**.

See also **include()**, **require_once()**, **include_once()**, **readfile()**, and **virtual()**.

include()

The **include()** statement includes and evaluates the specified file.

If "URL fopen wrappers" are enabled in PHP (which they are in the default configuration), you can specify the file to be **include()**ed using an URL instead of a local pathname. See [Remote files](#) and **fopen()** for more information.

An important note about how this works is that when a file is **include()**ed or **require()**ed, parsing drops out of PHP mode and into HTML mode at the beginning of the target file, and resumes again at the end. For this reason, any code inside the target file which should be executed as PHP code must be enclosed within [valid PHP start and end tags](#).

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 files.

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($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);
}
```

In both PHP 3 and PHP 4, it is possible to execute a **return** statement inside an **include()**ed file, in order to terminate processing in that file and return to the script which called it. Some differences in the way this works exist, however. The first is that in PHP 3, the **return** may not appear inside a block unless it's a function block, in which case the **return** applies to that function and not the whole file. In PHP 4, however, this restriction does not exist. Also, PHP 4 allows you to return values from **include()**ed files. You can take the value of the **include()** call as you would a normal function. This generates a parse error in PHP 3.

Example 11-1. include() in PHP 3 and PHP 4

Assume the existence of the following file (named `test.inc`) in the same directory as the main file:

```
<?php
echo "Before the return <br>\n";
if (1) {
```

```

        return 27;
}
echo "After the return <br>\n";
?>
```

Assume that the main file (`main.html`) contains the following:

```
<?php
$retval = include ('test.inc');
echo "File returned: '$retval'<br>\n";
?>
```

When `main.html` is called in PHP 3, it will generate a parse error on line 2; you can't take the value of an `include()` in PHP 3. In PHP 4, however, the result will be:

```
Before the return
File returned: '27'
```

Now, assume that `main.html` has been altered to contain the following:

```
<?php
include ('test.inc');
echo "Back in main.html<br>\n";
?>
```

In PHP 4, the output will be:

```
Before the return
Back in main.html
```

However, PHP 3 will give the following output:

```
Before the return
27Back in main.html
```

```
Parse error: parse error in /home/torben/public_html/phptest/main.html on line 5
```

The above parse error is a result of the fact that the `return` statement is enclosed in a non-function block within `test.inc`. When the `return` is moved outside of the block, the output is:

```
Before the return
27Back in main.html
```

The spurious '27' is due to the fact that PHP 3 does not support returning values from files like that.

When a file is `include()`d, the code it contains inherits the variable scope of the line on which the `include()` occurs. Any variables available at that line in the calling file will be available within the called file. If the `include()` occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function.

If the `include()`d file is called via HTTP using the `fopen` wrappers, and if the target server interprets the target file as PHP code, variables may be passed to the `include()`d file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as `include()`ing the file and having it inherit the parent file's variable scope; the script is actually being run on the remote server and the result is then being included into the local script.

```
/* This example assumes that someserver is configured to parse .php
 * files and not .txt files. Also, 'works' here means that the variables
 * $varone and $vartwo are available within the include()ed file. */

/* Won't work; file.txt wasn't handled by someserver. */
```

```

include ("http://someserver/file.txt?varone=1&vartwo=2");

/* Won't work; looks for a file named 'file.php?varone=1&vartwo=2'
 * on the local filesystem. */
include ("file.php?varone=1&vartwo=2");

/* Works. */
include ("http://someserver/file.php?varone=1&vartwo=2");

$varone = 1;
$vartwo = 2;
include ("file.txt"); /* Works. */
include ("file.php"); /* Works. */

```

See also **require()**, **require_once()**, **include_once()**, **readfile()**, and **virtual()**.

require_once()

The **require_once()** statement replaces itself with the specified file, much like the C preprocessor's #include works, and in that respect is similar to the **require()** statement. The main difference is that in an inclusion chain, the use of **require_once()** will assure that the code is added to your script only once, and avoid clashes with variable values or function names that can happen.

For example, if you create the following 2 include files `utils.inc` and `foolib.inc`

Example 11-2. utils.inc

```

<?php
define(PHPVERSION, floor(phpversion()));
echo "GLOBALS ARE NICE\n";
function goodTea() {
    return "Oolong tea tastes good!";
}
?>

```

Example 11-3. foolib.inc

```

<?php
require ("utils.inc");
function showVar($var) {
    if (PHPVERSION == 4) {
        print_r($var);
    } else {
        var_dump($var);
    }
}

// bunch of other functions ...
?>

```

And then you write a script `cause_error_require.php`

Example 11-4. cause_error_require.php

```

<?php
require("foolib.inc");

```

```
/* the following will generate an error */
require("utils.inc");
$foo = array("1",array("complex","quaternion"));
echo "this is requiring utils.inc again which is also\n";
echo "required in foolib.inc\n";
echo "Running goodTea: ".goodTea()."\n";
echo "Printing foo: \n";
showVar($foo);
?>
```

When you try running the latter one, the resulting output will be (using PHP 4.0.1pl2):

```
GLOBALS ARE NICE
GLOBALS ARE NICE

Fatal error: Cannot redeclare goodTea() in utils.inc on line 5
```

By modifying `foolib.inc` and `cause_error_require.php` to use `require_once()` instead of `require()` and renaming the last one to `avoid_error_require_once.php`, we have:

Example 11-5. foolib.inc (fixed)

```
...
require_once("utils.inc");
function showVar($var) {
...
}
```

Example 11-6. avoid_error_require_once.php

```
...
require_once("foolib.inc");
require_once("utils.inc");
$foo = array("1",array("complex","quaternion"));
...

```

And when running the latter, the output will be (using PHP 4.0.1pl2):

```
GLOBALS ARE NICE
this is requiring globals.inc again which is also
required in foolib.inc
Running goodTea: Oolong tea tastes good!
Printing foo:
Array
(
    [0] => 1
    [1] => Array
        (
            [0] => complex
            [1] => quaternion
        )
)
```

Also note that, analogous to the behavior of the `#include` of the C preprocessor, this statement acts at "compile time", e.g. when the script is parsed and before it is executed, and should not be used for parts of the script that need to be inserted dynamically during its execution. You should use `include_once()` or `include()` for that purpose.

For more examples on using **require_once()** and **include_once()**, look at the PEAR code included in the latest PHP source code distributions.

See also: **require()**, **include()**, **include_once()**, **get_required_files()**, **get_included_files()**, **readfile()**, and **virtual()**.

include_once()

The **include_once()** statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the **include()** statement, with the important difference that if the code from a file has already been included, it will not be included again.

As mentioned in the **require_once()** description, the **include_once()** should be used in the cases in which the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassessments, etc.

For more examples on using **require_once()** and **include_once()**, look at the PEAR code included in the latest PHP source code distributions.

include_once() was added in PHP 4.0.1pl2

See also: **require()**, **include()**, **require_once()**, **get_required_files()**, **get_included_files()**, **readfile()**, and **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.

In PHP 3, functions must be defined before they are referenced. No such requirement exists in PHP 4.

PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

PHP 3 does not support variable numbers of arguments to functions, although default arguments are supported (see [Default argument values](#) for more information). PHP 4 supports both: see [Variable-length argument lists](#) and the function references for [func_num_args\(\)](#), [func_get_arg\(\)](#), and [func_get_args\(\)](#) for more information.

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 supported only in PHP 4 and later; see [Variable-length argument lists](#) and the function references for [func_num_args\(\)](#), [func_get_arg\(\)](#), and [func_get_args\(\)](#) for more information. A similar effect can be achieved in PHP 3 by passing an array of arguments to a function:

```
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.

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.
```

Variable-length argument lists

PHP 4 has support for variable-length argument lists in user-defined functions. This is really quite easy, using the `func_num_args()`, `func_get_arg()`, and `func_get_args()` functions.

No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

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();
```

To return a reference from a function, you have to use the reference operator & in both the function declaration and when assigning the returned value to a variable:

```
function &returns_reference() {
    return $someref;
}

$newref =&returns_reference();
```

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->PHP 3 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 PHP 3 form) to call the `old_function`.

Variable functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Example 12-1. Variable function example

```
<?php
function foo() {
    echo "In foo()  
\n";
}

function bar( $arg = " " ) {
    echo "In bar(); argument was '$arg'.  
\n";
}

$func = 'foo';
$func();
$func = 'bar';
$func( 'test' );
?>
```

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.

Note: In PHP 4, only constant initializers for `var` variables are allowed. Use constructors for non-constant initializers.

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. Multiple inheritance is not supported.

```
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 carts 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 extends 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.

Chapter 14. References Explained

What References Are

References are a means in PHP to access the same variable content by different names. They are not like C pointers, they are symbol table aliases. Note that in PHP, variable name and variable content are different, so the same content can have different names. The most close analogy is with Unix filenames and files - variable names are directory entries, while variable contents is the file itself. References can be thought of as hardlinking in Unix filesystem.

What References Do

PHP references allow you to make two variables to refer to the same content. Meaning, when you do:

```
$a =& $b
```

it means that `$a` and `$b` point to the same variable.

Note: `$a` and `$b` are completely equal here, that's not `$a` is pointing to `$b` or vice versa, that's `$a` and `$b` pointing to the same place.

The same syntax can be used with functions, that return references, and with new operator (in PHP 4.0.4 and later):

```
$bar = & new fooclass();
$foo = & find_var ($bar);
```

Note: Unless you use the syntax above, the result of `$bar = new fooclass()` will not be the same variable as `$this` in the constructor, meaning that if you have used reference to `$this` in the constructor, you should use reference assignment, or you get two different objects.

The second thing references do is to pass variables by-reference. This is done by making a local variable in a function and a variable in the calling scope reference to the same content. Example:

```
function foo (&$var) {
    $var++;
}

$a=5;
foo ($a);
```

will make `$a` to be 6. This happens because in the function `foo` the variable `$var` refers to the same content as `$a`. See also more detailed explanations about [passing by reference](#).

The third thing reference can do is [return by reference](#).

What References Are Not

As said before, references aren't pointers. That means, the following construct won't do what you expect:

```
function foo (&$var) {
    $var = & $GLOBALS[ "baz" ];
}
foo($bar);
```

What happens is that `$var` in `foo` will be bound with `$bar` in caller, but then it will be re-bound with `$GLOBALS["baz"]`. There's no way to bind `$bar` in the calling scope to something else using the reference mechanism, since `$bar` is not available in the function `foo` (it is represented by `$var`, but `$var` has only variable contents and not name-to-value binding in the calling symbol table).

Passing by Reference

You can pass variable to function by reference, so that function could modify its arguments. The syntax is as follows:

```
function foo (&$var) {
    $var++;
}

$a=5;
foo ($a);
// $a is 6 here
```

Note that there's no reference sign on function call - only on function definition. Function definition alone is enough to correctly pass the argument by reference.

Following things can be passed by reference:

- Variable, i.e. `foo($a)`
- New statement, i.e. `foo(new foobar())`
- Reference, returned from a function, i.e.:

```
function &bar()
{
    $a = 5;
    return $a;
}
foo(bar());
```

See also explanations about [returning by reference](#).

Any other expression should not be passed by reference, as the result is undefined. For example, the following examples of passing by reference are invalid:

```
function bar() // Note the missing &
{
    $a = 5;
    return $a;
}
foo(bar());

foo($a = 5) // Expression, not variable
foo(5) // Constant, not variable
```

These requirements are for PHP 4.0.4 and later.

Returning References

Returning by-reference is useful when you want to use a function to find which variable a reference should be bound to. When returning references, use this syntax:

```
function &find_var ($param) {
    ...code...
    return $found_var;
}

$foo =& find_var ($bar);
$foo->x = 2;
```

In this example, the property of the object returned by the `find_var` function would be set, not the copy, as it would be without using reference syntax.

Note: Unlike parameter passing, here you have to use `&` in both places - to indicate that you return by-reference, not a copy as usual, and to indicate that reference binding, rather than usual assignment, should be done for `$foo`.

Unsetting References

When you unset the reference, you just break the binding between variable name and variable content. This does not mean that variable content will be destroyed. For example:

```
$a = 1;
$b =& $a;
unset ($a);
```

won't unset `$b`, just `$a`.

Again, it might be useful to think about this as analogous to Unix **unlink** call.

Spotting References

Many syntax constructs in PHP are implemented via referencing mechanisms, so everything told above about reference binding also apply to these constructs. Some constructs, like passing and returning by-reference, are mentioned above. Other constructs that use references are:

global References

When you declare variable as **global \$var** you are in fact creating reference to a global variable. That means, this is the same as:

```
$var =& $GLOBALS[ "var" ];
```

That means, for example, that unsetting `$var` won't unset global variable.

\$this

In an object method, `$this` is always reference to the caller object.

Part III. Features

Chapter 15. Error Handling

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

Table 15-1. PHP error types

Value	Constant	Description	Note
1	E_ERROR	fatal run-time errors	
2	E_WARNING	run-time warnings (non fatal errors)	
4	E_PARSE	compile-time parse errors	
8	E_NOTICE	run-time notices (less serious than warnings)	
16	E_CORE_ERROR	fatal errors that occur during PHP's initial startup	PHP 4 only
32	E_CORE_WARNING	warnings (non fatal errors) that occur during PHP's initial startup	PHP 4 only
64	E_COMPILE_ERROR	fatal compile-time errors	PHP 4 only
128	E_COMPILE_WARNING	compile-time warnings (non fatal errors)	PHP 4 only
256	E_USER_ERROR	user-generated error message	PHP 4 only
512	E_USER_WARNING	user-generated warning message	PHP 4 only
1024	E_USER_NOTICE	user-generated notice message	PHP 4 only
	E_ALL	all of the above, as supported	

The above values (either numerical or symbolic) are used to build up a bitmask that specifies which errors to report. You can use the [bitwise operators](#) to combine these values or mask out certain types of errors. Note that only '|', '~', '!', and '&' will be understood within `php.ini`, however, and that no bitwise operators will be understood within `php3.ini`.

In PHP 4, the default [error_reporting](#) setting is `E_ALL & ~E_NOTICE`, meaning to display all errors and warnings which are not E_NOTICE-level. In PHP 3, the default setting is `(E_ERROR | E_WARNING | E_PARSE)`, meaning the same thing. Note, however, that since constants are not supported in PHP 3's `php3.ini`, the [error_reporting](#) setting there must be numeric; hence, it is 7.

The initial setting can be changed in the ini file with the [error_reporting](#) directive, in your Apache `httpd.conf` file with the `php_error_reporting` (`php3_error_reporting` for PHP 3) directive, and lastly it may be set at runtime within a script by using the [error_reporting\(\)](#) function.

Warning

When upgrading code or servers from PHP 3 to PHP 4 you should check these settings and calls to [error_reporting\(\)](#) or you might disable reporting the new error types, especially `E_COMPILE_ERROR`. This may lead to empty documents without any feedback of what happened or where to look for the problem.

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`.

Note: The [@ error-control operator](#) prefix will not disable messages that are the result of parse errors.

Warning

Currently the `@ error-control operator` prefix will even disable error reporting for critical errors that will terminate script execution. Among other things, this means that if you use `@` to suppress errors from a certain function and either it isn't available or has been mistyped, the script will die right there with no indication as to why.

Below we can see an example of using the error handling capabilities in PHP. We define a error handling function which logs the information into a file (using an XML format), and e-mails the developer in case a critical error in the logic happens.

Example 15-1. Using error handling in a script

```

<?php
// we will do our own error handling
error_reporting(0);

// user defined error handling function
function userErrorHandler ($errno, $errmsg, $filename, $linenum, $vars) {
    // timestamp for the error entry
    $dt = date("Y-m-d H:i:s (T)");

    // define an assoc array of error string
    // in reality the only entries we should
    // consider are 2,8,256,512 and 1024
    $errortype = array (
        1    => "Error",
        2    => "Warning",
        4    => "Parsing Error",
        8    => "Notice",
        16   => "Core Error",
        32   => "Core Warning",
        64   => "Compile Error",
        128  => "Compile Warning",
        256  => "User Error",
        512  => "User Warning",
        1024=> "User Notice"
    );
    // set of errors for which a var trace will be saved
    $user_errors = array(E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE);

    $err = "<erroreentry>\n";
    $err .= "\t<datetime>".$dt."</datetime>\n";
    $err .= "\t<errnum>".$errno."</errnumber>\n";
    $err .= "\t<errortype>".$errortype[$errno]."</errortype>\n";
    $err .= "\t<errmsg>".$errmsg."</errmsg>\n";
    $err .= "\t<scriptname>".$filename."</scriptname>\n";
    $err .= "\t<scriptlinenum>".$linenum."</scriptlinenum>\n";

    if (in_array($errno, $user_errors))
        $err .= "\t<vartrace>".wddx_serialize_value($vars, "Variables")."</vartrace>\n";
    $err .= "</erroreentry>\n\n";

    // for testing
    // echo $err;

    // save to the error log, and e-mail me if there is a critical user error
    error_log($err, 3, "/usr/local/php4/error.log");
    if ($errno == E_USER_ERROR)
        mail("phpdev@mydomain.com", "Critical User Error", $err);
}

```

```

function distance ($vect1, $vect2) {
    if (!is_array($vect1) || !is_array($vect2)) {
        trigger_error("Incorrect parameters, arrays expected", E_USER_ERROR);
        return NULL;
    }

    if (count($vect1) != count($vect2)) {
        trigger_error("Vectors need to be of the same size", E_USER_ERROR);
        return NULL;
    }

    for ($i=0; $i<count($vect1); $i++) {
        $c1 = $vect1[$i]; $c2 = $vect2[$i];
        $d = 0.0;
        if (!is_numeric($c1)) {
            trigger_error("Coordinate $i in vector 1 is not a number, using zero",
                          E_USER_WARNING);
            $c1 = 0.0;
        }
        if (!is_numeric($c2)) {
            trigger_error("Coordinate $i in vector 2 is not a number, using zero",
                          E_USER_WARNING);
            $c2 = 0.0;
        }
        $d += $c2*$c2 - $c1*$c1;
    }
    return sqrt($d);
}

$old_error_handler = set_error_handler("userErrorHandler");

// undefined constant, generates a warning
$t = I_AM_NOT_DEFINED;

// define some "vectors"
$a = array(2,3,"foo");
$b = array(5.5, 4.3, -1.6);
$c = array (1,-3);

// generate a user error
$t1 = distance($c,$b)."\n";

// generate another user error
$t2 = distance($b,"i am not an array")."\n";

// generate a warning
$t3 = distance($a,$b)."\n";

?>

```

This is just a simple example showing how to use the [Error Handling and Logging functions](#).

See also **error_reporting()**, **error_log()**, **set_error_handler()**, **restore_error_handler()**, **trigger_error()**, **user_error()**

Chapter 16. Creating and manipulating images

PHP is not limited to creating just HTML output. It can also be used to create and manipulate image files in a variety of different image formats, including gif, png, jpg, wbmp, and xpm. Even more convenient, php can output image streams directly to a browser. You will need to compile PHP with the GD library of image functions for this to work. GD and PHP may also require other libraries, depending on which image formats you want to work with. GD stopped supporting Gif images in version 1.6.

Example 16-1. PNG creation with PHP

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

This example would be called from a page with a tag like: The above button.php script then takes this "text" string and overlays it on top of a base image which in this case is "images/button1.png" 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 17. 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 17-1. 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 Navigator 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 17-2. 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 access this resource\n";
    exit;
}

if(!isset($PHP_AUTH_USER) || ($SeenBefore == 1 && !strcmp($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. The user can press the ' _ ' key to clear their authentication information, however.

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 18. 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 19. 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 19-1. 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 PHP 3, the following variables will be defined within the destination script upon a successful upload, assuming that [register_globals](#) is turned on in `php3.ini`. If [track_vars](#) is turned on, they will also be available in PHP 3 within the global array `$HTTP_POST_VARS`. Note that the following variable names assume the use of the file upload name 'userfile', as used in the example above:

- `$userfile` - The temporary filename in which the uploaded file was stored on the server machine.
- `$userfile_name` - The original name or path 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"

In PHP 4, the behaviour is slightly different, in that the new global array `$HTTP_POST_FILES` is provided to contain the uploaded file information. This is still only available if [track_vars](#) is turned on, but [track_vars](#) is always turned on in versions of PHP after PHP 4.0.2.

The contents of `$HTTP_POST_FILES` are as follows. Note that this assumes the use of the file upload name 'userfile', as used in the example above:

```
$HTTP_POST_FILES['userfile']['name']
```

The original name of the file on the client machine.

```
$HTTP_POST_FILES['userfile']['type']
```

The mime type of the file, if the browser provided this information. An example would be "image/gif".

```
$HTTP_POST_FILES['userfile']['size']
```

The size, in bytes, of the uploaded file.

```
$HTTP_POST_FILES['userfile']['tmp_name']
```

The temporary filename of the file in which the uploaded file was stored on the server.

Files will by default be stored in the server's default temporary directory, unless another location has been given with the `upload_tmp_dir` directive in `php.ini`. The server's default directory 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. This environment variable can also be used to make sure that other operations are working on uploaded files, as well.

Example 19-2. Validating file uploads

The following examples are for versions of PHP 3 greater than 3.0.16, and versions of PHP 4 greater than 4.0.2. See the function entries for `is_uploaded_file()` and `move_uploaded_file()`.

```
<?php
if (is_uploaded_file($userfile)) {
    copy($userfile, "/place/to/put/uploaded/file");
} else {
    echo "Possible file upload attack: filename '$userfile'.";
}
/* ...or... */
move_uploaded_file($userfile, "/place/to/put/uploaded/file");
?>
```

For earlier versions of PHP, you'll need to do something like the following.

Note: This will *not* work in versions of PHP 4 after 4.0.2. It depends on internal functionality of PHP which changed after that version.

```
<?php
/* Userland test for uploaded file. */
function is_uploaded_file($filename) {
    if (!$tmp_file = get_cfg_var('upload_tmp_dir')) {
        $tmp_file = dirname(tempnam(" ", " "));
    }
    $tmp_file .= '/' . basename($filename);
    /* User might have trailing slash in php.ini... */
    return (ereg_replace('/+', '/', $tmp_file) == $filename);
}

if (is_uploaded_file($userfile)) {
    copy($userfile, "/place/to/put/uploaded/file");
} else {
    echo "Possible file upload attack: filename '$userfile'.";
}
?>
```

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 `PHP 3.ini` file or the corresponding `php3_upload_max_filesize` Apache `.conf` directive. The default is 2 Megabytes.

Not validating which file you operate on may mean that users can access sensitive information in other directories.

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 19-3. Uploading multiple files

```
<form action="file-upload.php" 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_FILES (\$HTTP_POST_VARS in PHP 3)). 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.

\$userfile['name'][0], \$userfile['tmp_name'][0], \$userfile['size'][0], and \$userfile['type'][0] are also set.

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:

```
<?php 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 but 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 20. 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 (for versions up to 4.0.3) or set `allow_url_fopen` to off in `php.ini` (for newer versions), 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 20-1. 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 20-2. 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 21. 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 22. 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 is 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 PHP 4 has support for ISAPI, WSAPI, and NSAPI (on Windows), which 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. The behavior is essentially the same as for the multiprocess model described before. Note that SAPI support is not available in PHP 3.

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.

Note, however, that this can have some drawbacks if you are using a database with connection limits that are exceeded by persistent child connections. If your database has a limit of 16 simultaneous connections, and in the course of a busy server session, 17 child threads attempt to connect, one will not be able to. If there are bugs in your scripts which do not allow the connections to shut down (such as infinite loops), a database with only 32 connections may be rapidly swamped. Check your database documentation for information on handling abandoned or idle connections.

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!

Part IV. Function Reference

I. Apache-specific Functions

apache_lookup_uri (PHP 3>= 3.0.4, PHP 4)

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

```
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: `Apache_lookup_uri()` only works when PHP is installed as an Apache module.

apache_note (PHP 3>= 3.0.2, PHP 4)

Get and set apache request notes

```
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 (PHP 3, PHP 4)

Fetch all HTTP request headers

```
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 (PHP 3, PHP 4)

Perform an Apache sub-request

```
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.

ascii2ebcdic (PHP 3>= 3.0.17)

Translate string from ASCII to EBCDIC

```
int ascii2ebcdic (string ascii_str)
```

ascii2ebcdic() is an Apache-specific function which is available only on EBCDIC based operating systems (OS/390, BS2000). It translates the ASCII encoded string *ascii_str* to its equivalent EBCDIC representation (binary safe), and returns the result.

See also the reverse function **ebcdic2ascii()**

ebcdic2ascii (PHP 3>= 3.0.17)

Translate string from EBCDIC to ASCII

```
int ebcdic2ascii (string ebcdic_str)
```

ebcdic2ascii() is an Apache-specific function which is available only on EBCDIC based operating systems (OS/390, BS2000). It translates the EBCDIC encoded string *ebcdic_str* to its equivalent ASCII representation (binary safe), and returns the result.

See also the reverse function **ascii2ebcdic()**

II. Array Functions

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Simple and multi-dimensional arrays are supported, and may be either user created or created by another function. There are specific database handling functions for populating arrays from database queries, and several functions return arrays.

See also **is_array()**, **explode()**, **implode()**, **split()** and **join()**.

array (unknown)

Create an array

```
array array ([mixed ...])
```

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.

Syntax "index => values", separated by commas, define index and values. index may be of type string or numeric. When index is omitted, a integer index is automatically generated, starting at 0. If index is an integer, next generated index will be the biggest integer index + 1. Note that when two identical index are defined, the last overwrite the first.

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")
);
```

Example 2. Automatic index with Array()

```
$array = array( 1, 1, 1, 1, 1, 8=>1, 4=>1, 19, 3=>13);
print_r($array);
```

which will display :

```
Array
(
    [0] => 1
    [1] => 1
    [2] => 1
    [3] => 13
    [4] => 1
    [8] => 1
    [9] => 19
)
```

Note that index '3' is defined twice, and keep its final value of 13. Index 4 is defined after index 8, and next generated index (value 19) is 9, since biggest index was 8.

This example creates a 1-based array.

Example 3. 1-based index with Array()

```
$firstquarter = array(1 => 'January', 'February', 'March');
print_r($firstquarter);
```

which will display :

```
Array
(
    [1] => 'January'
    [2] => 'February'
    [3] => 'March'
)
```

See also: [list\(\)](#).

array_count_values (PHP 4 >= 4.0b4)

Counts all the values of an array

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

Array_count_values() returns an array using 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)
```

array_diff (PHP 4 >= 4.0.1)

Computes the difference of arrays

```
array array_diff (array array1, array array2 [, array ...])
```

Array_diff() returns an array containing all the values of *array1* that are not present in any of the other arguments. Note that keys are preserved.

Example 1. Array_diff() example

```
$array1 = array ("a" => "green", "red", "blue");
$array2 = array ("b" => "green", "yellow", "red");
$result = array_diff ($array1, $array2);
```

This makes \$result have array ("blue");

See also [array_intersect\(\)](#).

array_flip (PHP 4 >= 4.0b4)

Flip all the values of an array

```
array array_flip (array trans)
```

Array_flip() returns an array in flip order.

Example 1. Array_flip() example

```
$trans = array_flip ($trans);
$original = strtr ($str, $trans);
```

array_intersect (PHP 4 >= 4.0.1)

Computes the intersection of arrays

```
array array_intersect (array array1, array array2 [, array ...])
```

Array_intersect() returns an array containing all the values of *array1* that are present in all the arguments. Note that keys are preserved.

Example 1. Array_intersect() example

```
$array1 = array ("a" => "green", "red", "blue");
$array2 = array ("b" => "green", "yellow", "red");
$result = array_intersect ($array1, $array2);
```

This makes \$result have array ("a" => "green", "red");

See also **array_diff()**.

array_keys (PHP 4)

Return all the keys of an array

```
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")
```

```
$array = array ("blue", "red", "green", "blue", "blue");
array_keys ($array, "blue"); // returns array (0, 3, 4)

$array = array ("color" => array("blue", "red", "green"), "size" => ar-
ray("small", "medium", "large"));
array_keys ($array); // returns array ("color", "size")
```

Note: This function was added to PHP 4, below is an implementation for those still using PHP 3.

Example 2. Implementation of array_keys() for PHP 3 users

```
function array_keys ($arr, $term="") {
    $t = array();
    while (list($k,$v) = each($arr)) {
        if ($term && $v != $term)
            continue;
        $t[] = $k;
    }
    return $t;
}
```

See also **array_values()**.

array_merge (PHP 4)

Merge two or more arrays

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

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 have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays have the same numeric key, the later value will not overwrite the original value, but will be appended.

Example 1. array_merge() example

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

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

See also **array_merge_recursive()**.

array_merge_recursive (PHP 4 >= 4.0.1)

Merge two or more arrays recursively

```
array array_merge_recursive (array array1, array array2 [, array ...])
```

Array_merge_recursive() 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 have the same string keys, then the values for these keys are merged together into an array, and this is done recursively, so that if one of the values is an array itself, the function will merge it with a corresponding entry in another array too. If, however, the arrays have the same numeric key, the later value will not overwrite the original value, but will be appended.

Example 1. **Array_merge_recursive()** example

```
$ar1 = array ("color" => array ("favorite" => "red"), 5);
$ar2 = array (10, "color" => array ("favorite" => "green", "blue"));
$result = array_merge_recursive ($ar1, $ar2);
```

Resulting array will be array ("color" => array ("favorite" => array ("red", "green"), "blue"),
5, 10).

See also **array_merge()**.

array_multisort (PHP 4 >= 4.0b4)

Sort multiple or multi-dimensional arrays

```
bool array_multisort (array ar1 [, mixed arg [, mixed ... [, array ...]]])
```

Array_multisort() can be used to sort several arrays at once or a multi-dimensional array according by one or more dimensions. It maintains key association when sorting.

The input arrays are treated as columns of a table to be sorted by rows - this resembles the functionality of SQL ORDER BY clause. The first array is the primary one to sort by. The rows (values) in that array that compare the same are sorted by the next input array, and so on.

The argument structure of this function is a bit unusual, but flexible. The very first argument has to be an array. Subsequently, each argument can be either an array or a sorting flag from the following lists.

Sorting order flags:

- SORT_ASC - sort in ascending order
- SORT_DESC - sort in descending order

Sorting type flags:

- SORT_REGULAR - compare items normally
- SORT_NUMERIC - compare items numerically
- SORT_STRING - compare items as strings

No two sorting flags of the same type can be specified after each array. The sortings flags specified after an array argument apply only to that array - they are reset to default SORT_ASC and SORT_REGULAR after before each new array argument.

Returns true on success, false on failure.

Example 1. Sorting multiple arrays

```
$ar1 = array ("10", 100, 100, "a");
$ar2 = array (1, 3, "2", 1);
array_multisort ($ar1, $ar2);
```

In this example, after sorting, the first array will contain 10, "a", 100, 100. The second array will contain 1, 1, 2, "3". The entries in the second array corresponding to the identical entries in the first array (100 and 100) were sorted as well.

Example 2. Sorting multi-dimensional array

```
$ar = array (array ("10", 100, 100, "a"), array (1, 3, "2", 1));
array_multisort ($ar[0], SORT_ASC, SORT_STRING,
                 $ar[1], SORT_NUMERIC, SORT_DESC);
```

In this example, after sorting, the first array will contain 10, 100, 100, "a" (it was sorted as strings in ascending order), and the second one will contain 1, 3, "2", 1 (sorted as numbers, in descending order).

array_pad (PHP 4 >= 4.0b4)

Pad array to the specified length with a value

```
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 (PHP 4)

Pop the element off the end of array

```
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()**.

array_push (PHP 4)

Push one or more elements onto the end of array

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

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()**.

array_rand (PHP 4 >= 4.0.0)

Pick one or more random entries out of an array

```
mixed array_rand (array input [, int num_req])
```

Array_rand() is rather useful when you want to pick one or more random entries out of an array. It takes an *input* array and an optional argument *num_req* which specifies how many entries you want to pick - if not specified, it defaults to 1.

If you are picking only one entry, **array_rand()** returns the key for a random entry. Otherwise, it returns an array of keys for the random entries. This is done so that you can pick random keys as well as values out of the array.

Don't forget to call **rand()** to seed the random number generator.

Example 1. Array_rand() example

```
rand ((double) microtime() * 10000000);
$input = array ("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$rand_keys = array_rand ($input, 2);
print $input[$rand_keys[0]]."\n";
print $input[$rand_keys[1]]."\n";
```

array_reverse (PHP 4 >= 4.0b4)

Return an array with elements in reverse order

```
array array_reverse (array array [, bool preserve_keys])
```

Array_reverse() takes input *array* and returns a new array with the order of the elements reversed, preserving the keys if *preserve_keys* is true.

Example 1. Array_reverse() example

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

This makes *\$result* have array (array ("green", "red"), 4.0, "php"). But *\$result2[0]* is still "php".

Note: The second parameter was added in PHP 4.0.3.

array_shift (PHP 4)

Pop an element off the beginning of array

```
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()**.

array_slice (PHP 4)

Extract a slice of the array

```
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()**.

array_splice (PHP 4)

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

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

Array_splice() removes 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()`.

array_unique (PHP 4 >= 4.0.1)

Removes duplicate values from an array

```
array array_unique (array array)
```

Array_unique() takes input `array` and returns a new array without duplicate values. Note that keys are preserved.

Example 1. `Array_unique()` example

```
$input = array ("a" => "green", "red", "b" => "green", "blue", "red");
$result = array_unique ($input);
```

This makes `$result` have `array ("a" => "green", "red", "blue")`.

array_unshift (PHP 4)

Push one or more elements onto the beginning of array

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

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()**.

array_values (PHP 4)

Return all the values of an array

```
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 to PHP 4, below is an implementation for those still using PHP 3.

Example 2. Implementation of array_values() for PHP 3 users

```
function array_values ($arr) {
    $t = array();
    while (list($k, $v) = each ($arr)) {
        $t[] = $v;
        return $t;
    }
}
```

array_walk (PHP 3>= 3.0.3, PHP 4)

Apply a user function to every member of an array

```
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.

In PHP 4 **reset()** needs to be called as necessary since **array_walk()** does not reset the array by default.

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

See also **each()** and **list()**.

arsort (PHP 3, PHP 4)

Sort an array in reverse order and maintain index association

```
void arsort (array array [, int sort_flags])
```

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);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\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.

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see **sort()**.

See also: **asort()**, **rsort()**, **ksort()**, and **sort()**.

asort (PHP 3, PHP 4)

Sort an array and maintain index association

```
void asort (array array [, int sort_flags])
```

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);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\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.

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see **sort()**.

See also **arsort()**, **rsort()**, **ksort()**, and **sort()**.

compact (PHP 4)

Create array containing variables and their values

```
array compact (mixed varname [, mixed ...])
```

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.

Any strings that are not set will simply be skipped.

Example 1. Compact() example

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

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

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

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

See also **extract()**.

count (PHP 3, PHP 4)

Count elements in a variable

```
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.

Example 1. Count() example

```
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count ($a);
// $result == 3
```

See also: **sizeof()**, **isset()**, and **is_array()**.

current (PHP 3, PHP 4)

Return the current element in an array

```
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 (PHP 3, PHP 4)

Return the next key and value pair from an array

```
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>";
reset ($HTTP_POST_VARS);
while (list ($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 (PHP 3, PHP 4)

Set the internal pointer of an array to its last element

```
mixed end (array array)
```

End() advances *array*'s internal pointer to the last element, and returns that element.

See also: **current()**, **each()**, **end()**, **next()**, and **reset()**.

extract (PHP 3>= 3.0.7, PHP 4)

Import variables into the symbol table from an array

```
int 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.

Note: Since version 4.0.5 this function returns the number of variables extracted.

extract() checks each key to see whether it constitutes a valid variable name and also for collisions with existing variables in the symbol table. The way invalid/numeric keys and 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 variable name with *prefix*.

EXTR_PREFIX_ALL

Prefix all variable names with *prefix*. Since PHP 4.0.5 this includes numeric ones as well.

EXTR_PREFIX_INVALID

Only prefix invalid/numeric variable names with *prefix*. This flag has been added in PHP 4.0.5.

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, EXTR_PREFIX_ALL, or EXTR_PREFIX_INVALID. If the prefixed result is not a valid variable name, it is not imported into the symbol table.

extract() returns the number of variables successfully imported into the symbol table.

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

```
<?php

/* 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*.

You must use an associative array, a numerically indexed array will not produce results.

See also: **compact()**.

in_array (PHP 4)

Return true if a value exists in an array

```
bool in_array (mixed needle, array haystack, bool strict)
```

Searches *haystack* for *needle* and returns true if it is found in the array, false otherwise.

If the third parameter *strict* is set to TRUE then the **in_array()** will also check the types of the *needle* in the *haystack*.

Example 1. In_array() example

```
$os = array ("Mac", "NT", "Irix", "Linux");
if (in_array ("Irix", $os)){
    print "Got Irix";
}
```

Example 2. In_array() with strict example

```
// This will output:
1.13 found with strict check
```

key (PHP 3, PHP 4)

Fetch a key from an associative array

```
mixed key (array array)
```

Key() returns the index element of the current array position.

See also **current()** and **next()**.

krsort (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sort an array by key in reverse order

```
int krsort (array array [, int sort_flags])
```

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);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key -> $val\n";
}
```

This example would display:

```
fruits[d] = lemon
fruits[c] = apple
fruits[b] = banana
fruits[a] = orange
```

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see **sort()**.

See also **asort()**, **arsort()**, **ksort()** **sort()**, **natsort()** and **rsort()**.

ksort (PHP 3, PHP 4)

Sort an array by key

```
int ksort (array array [, int sort_flags])
```

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);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key -> $val\n";
}
```

This example would display:

```
fruits[a] = orange
fruits[b] = banana
fruits[c] = apple
fruits[d] = lemon
```

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see **sort()**.

See also **asort()**, **arsort()**, **sort()**, **natsort()**, and **rsort()**.

Note: The second parameter was added in PHP 4.

list (unknown)

Assign variables as if they were an array

```
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.php?id=$id\">$name</a></td>\n".
        "     <td>$salary</td>\n".
        "   </tr>\n");
}

?>

</table>
```

See also **each()** and **array()**.

natsort (PHP 4 >= 4.0RC2)

Sort an array using a "natural order" algorithm

```
void natsort (array array)
```

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would. This is described as a "natural ordering". An example of the difference between this algorithm and the regular computer string sorting algorithms (used in **sort()**) can be seen below:

Example 1. natsort() example

```
$array1 = $array2 = array ("img12.png", "img10.png", "img2.png", "img1.png");

sort($array1);
echo "Standard sorting\n";
print_r($array1);

natsort($array2);
echo "\nNatural order sorting\n";
print_r($array2);
```

The code above will generate the following output:

```
Standard sorting
Array
```

```

(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
)

Natural order sorting
Array
(
    [3] => img1.png
    [2] => img2.png
    [1] => img10.png
    [0] => img12.png
)

```

For more information see: Martin Pool's Natural Order String Comparison (<http://www.linuxcare.com.au/projects/natsort/>) page.

See also **natcasesort()**, **strnatcmp()** and **strnatcasecmp()**.

natcasesort (PHP 4 >= 4.0RC2)

Sort an array using a case insensitive "natural order" algorithm

```
void natcasesort (array array)
```

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would. This is described as a "natural ordering".

natcasesort() is a case insensitive version of **natsort()**. See **natsort()** for an example of the difference between this algorithm and the regular computer string sorting algorithms.

For more information see: Martin Pool's Natural Order String Comparison (<http://www.linuxcare.com.au/projects/natsort/>) page.

See also **sort()**, **natsort()**, **strnatcmp()** and **strnatcasecmp()**.

next (PHP 3, PHP 4)

Advance the internal array pointer of an array

```
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, or elements that have a key value of 0 then this function will return false for these elements as well. To properly traverse an array which may contain empty elements or elements with key values of 0 see the **each()** function.

See also: **current()**, **end()**, **prev()**, and **reset()**.

pos (PHP 3, PHP 4)

Get the current element from an array

```
mixed pos (array array)
```

This is an alias for **current()**.

See also: **end()**, **next()**, **prev()** and **reset()**.

prev (PHP 3, PHP 4)

Rewind the internal array pointer

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Create an array containing a range of integers

```
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 (PHP 3, PHP 4)

Set the internal pointer of an array to its first element

```
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()**, and **prev()**.

rsort (PHP 3, PHP 4)

Sort an array in reverse order

```
void rsort (array array [, int sort_flags])
```

This function sorts an array in reverse order (highest to lowest).

Example 1. Rsort() example

```
$fruits = array ("lemon", "orange", "banana", "apple");
rsort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key -> $val\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.

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see **sort()**.

See also: **arsort()**, **asort()**, **ksort()**, **sort()**, and **usort()**.

shuffle (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Shuffle an array

```
void shuffle (array array)
```

This function shuffles (randomizes the order of the elements in) an array. You must use **rand()** to seed this function.

Example 1. Shuffle() example

```
$numbers = range (1,20);
srand ((double)microtime()*1000000);
shuffle ($numbers);
while (list (, $number) = each ($numbers)) {
    echo "$number ";
}
```

See also **arsort()**, **asort()**, **ksort()**, **rsort()**, **sort()** and **usort()**.

sizeof (PHP 3, PHP 4)

Get the number of elements in an array

```
int sizeof (array array)
```

Returns the number of elements in the array.

See also **count()**.

sort (PHP 3, PHP 4)

Sort an array

```
void sort (array array [, int sort_flags])
```

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

Example 1. Sort() example

```
<?php

$fruits = array ("lemon", "orange", "banana", "apple");
sort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "fruits[" . $key . "] = " . $val;
}

?>
```

This example would display:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

The fruits have been sorted in alphabetical order.

The optional second parameter *sort_flags* may be used to modify the sorting behavior using these values:

Sorting type flags:

- **SORT_REGULAR** - compare items normally
- **SORT_NUMERIC** - compare items numerically
- **SORT_STRING** - compare items as strings

See also: **arsort()**, **asort()**, **ksort()**, **natsort()**, **natcasesort()**, **rsort()**, **usort()**, **array_multisort()**, and **uksort()**.

Note: The second parameter was added in PHP 4.

uasort (PHP 3>= 3.0.4, PHP 4)

Sort an array with a user-defined comparison function and maintain index association

```
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.

Note: Please see **usort()** and **uksort()** for examples of user-defined comparison functions.

See also: **usort()**, **uksort()**, **sort()**, **asort()**, **arsort()**, **ksort()** and **rsort()**.

uksort (PHP 3>= 3.0.4, PHP 4)

Sort an array by keys using a user-defined comparison function

```
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 cmp ($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}

$a = array (4 => "four", 3 => "three", 20 => "twenty", 10 => "ten");

uksort ($a, "cmp");

while (list ($key, $value) = each ($a)) {
    echo "$key: $value\n";
}
```

This example would display:

```
20: twenty
10: ten
4: four
```

```
3: three
```

See also: **usort()**, **uasort()**, **sort()**, **asort()**, **arsort()**, **ksort()**, **natsort()** and **rsort()**.

usort (PHP 3>= 3.0.3, PHP 4)

Sort an array by values using a user-defined comparison function

```
void usort (array array, string 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.

Example 2. Usort() example using multi-dimensional array

```
function cmp ($a, $b) {
    return strcmp($a["fruit"],$b["fruit"]);
}
```

```
$fruits[0]["fruit"] = "lemons";
$fruits[1]["fruit"] = "apples";
$fruits[2]["fruit"] = "grapes";

usort($fruits, "cmp");

while (list ($key, $value) = each ($fruits)) {
    echo "\$fruits[$key]: " . $value["fruit"] . "\n";
}
```

When sorting a multi-dimensional array, \$a and \$b contain references to the first index of the array.

This example would display:

```
$fruits[0]: apples
$fruits[1]: grapes
$fruits[2]: lemons
```

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: **uasort()**, **uksort()**, **sort()**, **asort()**, **arsort()**, **ksort()**, **natsort()**, and **rsort()**.

III. Aspell functions

The **aspell()** functions allows you to check the spelling on a word and offer suggestions.

Note: aspell works only with very old (up to .27.* or so) versions of aspell library. Neither this module, nor those versions of aspell library are supported any longer. If you want to use spell-checking capabilities in php, use [pspell](#) instead. It uses pspell library and works with newer versions of aspell.

You need the aspell library, available from: <http://aspell.sourceforge.net/>.

aspell_new (PHP 3>= 3.0.7, PHP 4)

Load a new dictionary

```
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 (PHP 3>= 3.0.7, PHP 4)

Check a word

```
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 (PHP 3>= 3.0.7, PHP 4)

Check a word without changing its case or trying to trim it

```
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, "test")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

aspell_suggest (PHP 3>= 3.0.7, PHP 4)

Suggest spellings of a word

```
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, "test")) {
    $suggestions=aspell_suggest ($aspell_link, "test");

    for ($i=0; $i < count ($suggestions); $i++) {
        echo "Possible spelling: " . $suggestions[$i] . "<br>";
    }
}
```

IV. BCMath Arbitrary Precision Mathematics Functions

These functions are only available if PHP was configured with `-enable-bcmath`.

Note: Due to changes in the licensing, the BCMATH library is distributed separate from the standard PHP source distribution. You can download the tar-gzipped archive at the url: <http://www.php.net/extra/number4.tar.gz>. Read the file `README.BCMATH` in the PHP distribution for more information.

bcadd (PHP 3, PHP 4)

Add two arbitrary precision numbers

```
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 (PHP 3, PHP 4)

Compare two arbitrary precision numbers

```
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.

bcdif (PHP 3, PHP 4)

Divide two arbitrary precision numbers

```
string bcdif (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 (PHP 3, PHP 4)

Get modulus of an arbitrary precision number

```
string bcmod (string left operand, string modulus)
```

Get the modulus of the *left operand* using *modulus*.

See also **bediv()**.

bcmul (PHP 3, PHP 4)

Multiply two arbitrary precision number

```
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 (PHP 3, PHP 4)

Raise an arbitrary precision number to another

```
string bcpow (string x, string y [, int scale])
```

Raise *x* to the power *y*. The optional *scale* can be used to set the number of digits after the decimal place in the result.

See also **bcsqrt()**.

bcscale (PHP 3, PHP 4)

Set default scale parameter for all bc math functions

```
string bcsetScale (int scale)
```

This function sets the default scale parameter for all subsequent bc math functions that do not explicitly specify a scale parameter.

bcsqrt (PHP 3, PHP 4)

Get the square root of an arbitrary precision number

```
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 (PHP 3, PHP 4)

Subtract one arbitrary precision number from another

```
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()**.

V. Calendar functions

The calendar functions are only available if you have compiled the calendar extension, found in either the "dl" or "ext" subdirectories of your PHP source code. Please see the README file before 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 (PHP 3, PHP 4)

Converts Julian Day Count to Gregorian date

```
string jdtogregorian (int julianday)
```

Converts Julian Day Count to a string containing the Gregorian date in the format of "month/day/year".

GregorianToJD (PHP 3, PHP 4)

Converts a Gregorian date to Julian Day Count

```
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 (PHP 3, PHP 4)

Converts a Julian Day Count to a Julian Calendar Date

```
string jdtojulian (int julianday)
```

Converts Julian Day Count to a string containing the Julian Calendar Date in the format of "month/day/year".

JulianToJD (PHP 3, PHP 4)

Converts a Julian Calendar date to Julian Day Count

```
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 (PHP 3, PHP 4)

Converts a Julian Day Count to the Jewish Calendar

```
string jdtojewish (int julianday)
```

Converts a Julian Day Count to the Jewish Calendar.

JewishToJD (PHP 3, PHP 4)

Converts a date in the Jewish Calendar to Julian Day Count

```
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 (PHP 3, PHP 4)

Converts a Julian Day Count to the French Republican Calendar

```
string jdtorfrench (int juliandaycount)
```

Converts a Julian Day Count to the French Republican Calendar.

FrenchToJD (PHP 3, PHP 4)

Converts a date from the French Republican Calendar to a Julian Day Count

```
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 (PHP 3, PHP 4)

Returns a month name

```
string jdmonthname (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 - abbreviated
1	Gregorian
2	Julian - abbreviated
3	Julian
4	Jewish
5	French Republican

JDDayOfWeek (PHP 3, PHP 4)

Returns the day of the week

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0RC2)

Get UNIX timestamp for midnight on Easter of a given year

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0RC2)

Get number of days after March 21 on which Easter falls for a given year

```
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()**.

unixtojd (PHP 4 >= 4.0RC2)

Convert UNIX timestamp to Julian Day

```
int unixtojd ([int timestamp])
```

Return the Julian Day for a UNIX *timestamp* (seconds since 1.1.1970), or for the current day if no *timestamp* is given.

See also **jdtounix()**.

Note: This function is only available in PHP versions after PHP 4RC1.

jdtounix (PHP 4 >= 4.0RC2)

Convert Julian Day to UNIX timestamp

```
int jdtounix (int jday)
```

This function will return a UNIX timestamp corresponding to the Julian Day given in *jday* or false if *jday* is not inside the UNIX epoch (Gregorian years between 1970 and 2037 or 2440588 <= *jday* <= 2465342)

See also **jdtounix()**.

Note: This function is only available in PHP versions after PHP 4RC1.

VI. CCVS API Functions

These functions interface the CCVS API, allowing you to directly work with CCVS from your PHP scripts. CCVS is RedHat's (<http://www.redhat.com/>) solution to the "middle-man" in credit card processing. It lets you directly address the credit card clearing houses via your *nix box and a modem. Using the CCVS module for PHP, you can process credit cards directly through CCVS via your PHP Scripts. The following references will outline the process.

To enable CCVS Support in PHP, first verify your CCVS installation directory. You will then need to configure PHP with the `-with-ccvs` option. If you use this option without specifying the path to your CCVS installation, PHP Will attempt to look in the default CCVS Install location (`/usr/local/ccvs`). If CCVS is in a non-standard location, run configure with: `-with-ccvs=$ccvs_path`, where `$ccvs_path` is the path to your CCVS installation. Please note that CCVS support requires that `$ccvs_path/lib` and `$ccvs_path/include` exist, and include `cv_api.h` under the include directory and `libccvs.a` under the lib directory.

Additionally, a `ccvsd` process will need to be running for the configurations you intend to use in your PHP scripts. You will also need to make sure the PHP Processes are running under the same user as your CCVS was installed as (e.g. if you installed CCVS as user '`ccvs`', your PHP processes must run as '`ccvs`' as well.)

Additional information about CCVS can be found at <http://www.redhat.com/products/ccvs>.

This documentation section is being worked on. Until then, RedHat maintains slightly outdated but still useful documentation at <http://www.redhat.com/products/ccvs/support/CCVS3.3docs/ProgPHP.html>.

(unknown)

()

VII. COM support functions for Windows

These functions are only available on the Windows version of PHP. These functions have been added in PHP 4.

com_load (PHP 3>= 3.0.3, PHP 4)

Creates a new reference to a COM component

```
string com_load (string module name [, string server name])
```

com_load() creates a new COM component and returns a reference to it. Returns `false` on failure.

com_invoke (PHP 3>= 3.0.3, PHP 4)

Calls a COM component's method.

```
mixed com_invoke (resource com_object, string function_name [, mixed function parameters, ...])
```

Com_invoke() invokes a method of the COM component referenced by *com_object*. Returns `false` on error, returns the *function_name*'s return value on success.

com_propget (PHP 3>= 3.0.3, PHP 4)

Gets the value of a COM Component's property

```
mixed com_propget (resource com_object, string property)
```

This function is an alias for **com_get()**.

com_get (PHP 3>= 3.0.3, PHP 4)

Gets the value of a COM Component's property

```
mixed com_get (resource com_object, string property)
```

Returns the value of the *property* of the COM component referenced by *com_object*. Returns `false` on error.

com_propput (PHP 3>= 3.0.3, PHP 4)

Assigns a value to a COM component's property

```
void com_propput (resource com_object, string property, mixed value)
```

This function is an alias for **com_set()**.

com_propset (PHP 3>= 3.0.3, PHP 4)

Assigns a value to a COM component's property

```
void com_propset (resource com_object, string property, mixed value)
```

This function is an alias for **com_set()**.

com_set (PHP 3>= 3.0.3, PHP 4)

Assigns a value to a COM component's property

```
void com_set (resource com_object, string property, mixed value)
```

Sets the value of the *property* of the COM component referenced by *com_object*. Returns **true** if *property* is set. Returns **false** on error.

VIII. Class/Object Functions

Introduction

About

These functions allow you to obtain information about classes and instance objects. You can obtain the name of the class to which a object belongs, as well as its member properties and methods. Using these functions, you can find out not only the class membership of an object, but also its parentage (i.e. what class is the object class extending).

An example of use

In this example, we first define a base class and an extension of the class. The base class describes a general vegetable, whether it is edible or not and what is its color. The subclass Spinach adds a method to cook it and another to find out if it is cooked.

Example 1. classes.inc

```
<?php

// base class with member properties and methods
class Vegetable {

    var $edible;
    var $color;

    function Vegetable( $edible, $color="green" ) {
        $this->edible = $edible;
        $this->color = $color;
    }

    function is_edible() {
        return $this->edible;
    }

    function what_color() {
        return $this->color;
    }
}

} // end of class Vegetable

// extends the base class
class Spinach extends Vegetable {

    var $cooked = false;

    function Spinach() {
        $this->Vegetable( true, "green" );
    }

    function cook_it() {
        $this->cooked = true;
    }

    function is_cooked() {
        return $this->cooked;
}
```

```

    }

} // end of class Spinach

?>

```

We then instantiate 2 objects from these classes and print out information about them, including their class parentage. We also define some utility functions, mainly to have a nice printout of the variables.

Example 2. test_script.php

```

<pre>
<?php

include "classes.inc";

// utility functions

function print_vars($obj) {
    $arr = get_object_vars($obj);
    while (list($prop, $val) = each($arr))
        echo "\t$prop = $val\n";
}

function print_methods($obj) {
    $arr = get_class_methods(get_class($obj));
    foreach ($arr as $method)
        echo "\tfunction $method()\n";
}

function class_parentage($obj, $class) {
    global $$obj;
    if (is_subclass_of($$obj, $class)) {
        echo "Object $obj belongs to class ".get_class($$obj);
        echo " a subclass of $class\n";
    } else {
        echo "Object $obj does not belong to a subclass of $class\n";
    }
}

// instantiate 2 objects

$veggie = new Vegetable(true, "blue");
$leafy = new Spinach();

// print out information about objects
echo "veggie: CLASS ".get_class($veggie)."\n";
echo "leafy: CLASS ".get_class($leafy);
echo ", PARENT ".get_parent_class($leafy)."\n";

// show veggie properties
echo "\nveggie: Properties\n";
print_vars($veggie);

// and leafy methods
echo "\nleafy: Methods\n";
print_methods($leafy);

echo "\nParentage:\n";

```

```
class_parentage("leafy", "Spinach");
class_parentage("leafy", "Vegetable");
?>
</pre>
```

One important thing to note in the example above is that the object \$leafy is an instance of the class Spinach which is a subclass of Vegetable, therefore the last part of the script above will output:

```
[...]
Parentage:
Object leafy does not belong to a subclass of Spinach
Object leafy belongs to class spinach a subclass of Vegetable
```


call_user_method (PHP 3>= 3.0.3, PHP 4)

Call a user method on an specific object

```
mixed call_user_method (string method_name, object obj [, mixed parameter [, mixed ...]])
```

Calls a the method referred by *method_name* from the user defined *obj* object. An example of usage is below, where we define a class, instantiate an object and use **call_user_method()** to call indirectly its **print_info** method.

```
<?php
class Country {
    var $NAME;
    var $TLD;

    function Country($name, $tld) {
        $this->NAME = $name;
        $this->TLD = $tld;
    }

    function print_info($prestr="") {
        echo $prestr."Country: ".$this->NAME."\n";
        echo $prestr."Top Level Domain: ".$this->TLD."\n";
    }
}

$ctry = new Country("Peru", "pe");

echo "* Calling the object method directly\n";
$ctry->print_info();

echo "\n* Calling the same method indirectly\n";
call_user_method ("print_info", $ctry, "\t");
?>
```

See also **call_user_func()**.

class_exists (PHP 4 >= 4.0b4)

Checks if the class has been defined

```
bool class_exists (string class_name)
```

This function returns true if the class given by *class_name* has been defined, false otherwise.

get_class (PHP 4 >= 4.0b2)

Returns the name of the class of an object

```
string get_class (object obj)
```

This function returns the name of the class of which the object *obj* is an instance.

See also **get_parent_class()**, **is_subclass_of()**

get_class_methods (PHP 4 >= 4.0RC1)

Returns an array of class methods' names

```
array get_class_methods (string class_name)
```

This function returns an array of method names defined for the class specified by *class_name*.

See also **get_class_vars()**, **get_object_vars()**

get_class_vars (PHP 4 >= 4.0RC1)

Returns an array of default properties of the class

```
array get_class_vars (string class_name)
```

This function will return an array of default properties of the class.

See also **get_class_methods()**, **get_object_vars()**

get_declared_classes (PHP 4 >= 4.0RC2)

Returns an array with the name of the defined classes

```
array get_declared_classes (void)
```

This function returns an array of the names of the declared classes in the current script.

Note: In PHP 4.0.1pl2, three extra classes are returned at the beginning of the array: `stdClass` (defined in `Zend/zend.c`), `OverloadedTestClass` (defined in `ext/standard/basic_functions.c`) and `Directory` (defined in `ext/standard/dir.c`).

get_object_vars (PHP 4 >= 4.0RC1)

Returns an associative array of object properties

```
array get_object_vars (object obj)
```

This function returns an associative array of defined object properties for the specified object *obj*. If variables declared in the class of which the *obj* is an instance, have not been assigned a value, those will not be returned in the array.

Example 1. Use of get_object_vars()

```
<?php
class Point2D {
    var $x, $y;
```

```

var $label;

function Point2D($x, $y) {
    $this->x = $x;
    $this->y = $y;
}

function setLabel($label) {
    $this->label = $label;
}

function getPoint() {
    return array("x" => $this->x,
                "y" => $this->y,
                "label" => $this->label);
}

$p1 = new Point2D(1.233, 3.445);
print_r(get_object_vars($p1));
// "$label" is declared but not defined
// Array
// (
//     [x] => 1.233
//     [y] => 3.445
// )

$p1->setLabel("point #1");
print_r(get_object_vars($p1));
// Array
// (
//     [x] => 1.233
//     [y] => 3.445
//     [label] => point #1
// )

?>
```

See also **get_class_methods()**, **get_class_vars()**

get_parent_class (PHP 4 >= 4.0b2)

Returns the name of the parent class of an object

```
string get_parent_class (object obj)
```

This function returns the name of the parent class to the class of which the object *obj* is an instance.

See also **get_class()**, **is_subclass_of()**

is_subclass_of (PHP 4 >= 4.0b4)

Determines if an object belongs to a subclass of the specified class

```
bool is_subclass_of (object obj, string superclass)
```

This function returns true if the object *obj*, belongs to a class which is a subclass of *superclass*, false otherwise.

See also **get_class()**, **get_parent_class()**

method_exists (PHP 4 >= 4.0b2)

Checks if the class method exists

```
bool method_exists (object object, string method_name)
```

This function returns true if the method given by *method_name* has been defined for the given *object*, false otherwise.

IX. ClibPDF functions

ClibPDF lets you create PDF documents with PHP. It is available at FastIO (<http://www.fastio.com/>) but it isn't 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's pdflib but, according to FastIO, ClibPDF is 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 shows no difference in speed at all. The file size is also similar if compression is turned off. So, try them both and see which one does the job for you.

This documentation should be read alongside the ClibPDF manual since it explains the library in much greater detail.

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 the handle for the document as their first parameter.

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 can't 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.

Note: The function **cpdf_set_font()** has changed since PHP 3 to support asian fonts. The encoding parameter is no longer an integer but a string.

One big advantage of ClibPDF over pdflib used to be 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()**.

Another nice feature of ClibPDF is the fact that any page can be modified at any time even if a new page has been already opened. The function **cpdf_set_current_page()** allows to leave the current page and presume modifying an other page.

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 to 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($cpdf, 0, 0, 0, 1, "Page 1");
cpdf_begin_text($cpdf);
cpdf_set_font($cpdf, "Times-Roman", 30, "WinAnsiEncoding");
cpdf_set_text_rendering($cpdf, 1);
cpdf_text($cpdf, "Times Roman outlined", 50, 750);
cpdf_end_text($cpdf);
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);
    }

    $ltime = getdate();

    /* draw hour hand */
    cpdf_save($pdf);
    cpdf_rotate($pdf, -($ltime['minutes']/60.0) + $ltime['hours'] - 3.0) * 30.0);
    cpdf_moveto($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, -($ltime['seconds']/60.0) + $ltime['minutes'] - 15.0) * 6.0);
    cpdf_moveto($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, -($ltime['seconds'] - 15.0) * 6.0));
cpdf_moveto($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_global_set_document_limits (PHP 4 >= 4.0b4)

Sets document limits for any pdf document

```
void cpdf_global_set_document_limits (int maxpages, int maxfonts, int maximages, int
maxannotations, int maxobjects)
```

The **cpdf_global_set_document_limits()** function sets several document limits. This function has to be called before **cpdf_open()** to take effect. It sets the limits for any document open afterwards.

See also **cpdf_open()**.

cpdf_set_creator (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the creator field in the pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the title field of the pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the subject field of the pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the keywords field of the pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Opens a new pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Closes the pdf document

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Starts new page

```
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 coordinate 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 (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Ends page

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Ends document

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Outputs the pdf document in memory buffer

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Writes the pdf document into a file

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Sets current page

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Starts text section

```
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", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf)
?>
```

See also **cpdf_end_text()**.

cpdf_end_text (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Ends text section

```
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", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf)
?>
```

See also **cpdf_begin_text()**.

cpdf_show (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Output text at current position

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Output text at position

```
void cpdf_show_xy (int pdf document, string text, double x-coor, double y-coor [, int mode])
```

The **cpdf_show_xy()** function outputs the string *text* at position with coordinates (*x-coor*, *y-coor*).

The optional parameter *mode* determines the unit length. If it's 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.

Note: The function **cpdf_show_xy()** is identical to **cpdf_text()** without the optional parameters.

See also **cpdf_text()**.

cpdf_text (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Output text with parameters

```
void cpdf_text (int pdf document, string text, double x-coor, double y-coor [, int mode [, double orientation [, int alignmode]]])
```

The **cpdf_text()** function outputs the string *text* at position with coordinates (*x-coor*, *y-coor*).

The optional parameter *mode* determines the unit length. If it's 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. The optional parameter *orientation* is the rotation of the text in degree. The optional parameter *alignmode* determines how the text is aligned.

See the ClibPDF documentation for possible values.

See also **cpdf_show_xy()**.

cpdf_set_font (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Select the current font face and size

```
void cpdf_set_font (int pdf document, string font name, double size, string encoding)
```

The **cpdf_set_font()** function sets 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: "MacRomanEncoding", "MacExpertEncoding", "WinAnsiEncoding", and "NULL". "NULL" stands for the font's built-in encoding.

See the ClibPDF Manual for more information, especially how to support asian fonts.

cpdf_set_leading (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets distance between text lines

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Determines how text is rendered

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets horizontal scaling of text

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the text rise

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets the text matrix

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets text position

```
void cpdf_set_text_pos (int pdf_document, double x-coor, double y-coor [, int mode])
```

The **cpdf_set_text_pos()** function sets the position of text for the next **cpdf_show()** function call.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_show()**, **cpdf_text()**.

cpdf_set_char_spacing (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets character spacing

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets spacing between words

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Output text in next line

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Returns width of text in current font

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Saves current environment

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Restores formerly saved environment

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets origin of coordinate system

```
void cpdf_translate (int pdf document, double x-coor, double y-coor [, int mode])
```

The **cpdf_translate()** function set the origin of coordinate system to the point (*x-coor*, *y-coor*).

The optional parameter *mode* determines the unit length. If it's 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_scale (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets scaling

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets rotation

```
void cpdf_rotate (int pdf_document, double angle)
```

The **cpdf_rotate()** function set the rotation in degress to *angle*.

cpdf_setflat (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets flatness

```
void cpdf_setflat (int pdf_document, double value)
```

The **cpdf_setflat()** function set the flatness to a value between 0 and 100.

cpdf_setlinejoin (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets linejoin parameter

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets linecap parameter

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets miter limit

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets line width

```
void cpdf_setlinewidth (int pdf_document, double width)
```

The **cpdf_setlinewidth()** function set the line width to *width*.

cpdf_setdash (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets dash pattern

```
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_newpath (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Starts a new path

```
void cpdf_newpath (int pdf_document)
```

The **cpdf_newpath()** starts a new path on the document given by the *pdf_document* parameter.

cpdf_moveto (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets current point

```
void cpdf_moveto (int pdf_document, double x-coor, double y-coor [, int mode])
```

The **cpdf_moveto()** function set the current point to the coordinates *x-coor* and *y-coor*.

The optional parameter *mode* determines the unit length. If it's 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_rmoveto (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Sets current point

```
void cpdf_rmoveto (int pdf_document, double x-coor, double y-coor [, int mode])
```

The **cpdf_rmoveto()** function set the current point relative to the coordinates *x-coor* and *y-coor*.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_moveto()**.

cpdf_curveto (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draws a curve

```
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 optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_rlineto()**, **cpdf_linet()**.

cpdf_linet (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draws a line

```
void cpdf_linet (int pdf_document, double x-coor, double y-coor [, int mode])
```

The **cpdf_linet()** function draws a line from the current point to the point with coordinates (*x-coor*, *y-coor*).

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_rlinet (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Draws a line

```
void cpdf_rlinet (int pdf_document, double x-coor, double y-coor [, int mode])
```

The **cpdf_rlinet()** function draws a line from the current point to the relative point with coordinates (*x-coor*, *y-coor*).

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_circle (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draw a circle

```
void cpdf_circle (int pdf_document, double x-coor, double y-coor, double radius [, int mode])
```

The **cpdf_circle()** function draws a circle with center at point (*x-coor*, *y-coor*) and radius *radius*.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_arc()**.

cpdf_arc (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draws an arc

```
void cpdf_arc (int pdf_document, double x-coor, double y-coor, double radius, double start, double end [, int mode])
```

The **cpdf_arc()** function draws an arc with center at point (*x-coor*, *y-coor*) and radius *radius*, starting at angle *start* and ending at angle *end*.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_circle()**.

cpdf_rect (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draw a rectangle

```
void cpdf_rect (int pdf_document, double x-coor, double y-coor, double width, double height [, int mode])
```

The **cpdf_rect()** function draws a rectangle with its lower left corner at point (*x-coor*, *y-coor*). This width is set to *width*. This height is set to *height*.

The optional parameter *mode* determines the unit length. If it's 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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Close path

```
void cpdf_closepath (int pdf_document)
```

The **cpdf_closepath()** function closes the current path.

cpdf_stroke (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Draw line along path

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Close path and draw line along path

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Fill current path

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Fill and stroke current path

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Close, fill and stroke current path

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Clips to current path

```
void cpdf_clip (int pdf document)
```

The **cpdf_clip()** function clips all drawing to the current path.

cpdf_setgray_fill (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets filling color to gray value

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets drawing color to gray value

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets drawing and filling color to gray value

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets filling color to rgb color value

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets drawing color to rgb color value

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b4)

Sets drawing and filling color to rgb color value

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Adds bookmark for current page

```
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
$c�푸df = cpdf_open(0);
cpdf_page_init($ cpfdf, 1, 0, 595, 842);
cpdf_add_outline($ cpfdf, 0, 0, 0, 1, "Page 1");
// ...
// some drawing
// ...
cpdf_finalize($ cpfdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($ cpfdf);
cpdf_close($ cpfdf);
?>
```

cpdf_set_page_animation (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Sets duration between pages

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Opens a JPEG image

```
int cpdf_import_jpeg (int pdf_document, string file_name, double x-coor, double y-coor,
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-coor*, *y-coor*). The image is rotated by *angle* degrees.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_place_inline_image()**.

cpdf_place_inline_image (PHP 3>= 3.0.9, PHP 4 >= 4.0b4)

Places an image on the page

```
void cpdf_place_inline_image (int pdf_document, int image, double x-coor, double y-coor,
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 position (*x-coor*, *y-coor*). The image can be scaled at the same time.

The optional parameter *mode* determines the unit length. If it's 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.

See also **cpdf_import_jpeg()**.

cpdf_add_annotation (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

Adds annotation

```
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 optional parameter *mode* determines the unit length. If it's 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.

X. CURL, Client URL Library Functions

PHP supports libcurl, a library, created by Daniel Stenberg, that allows you to connect and communicate to many different types of servers with many different types of protocols. libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols. libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading (this can also be done with PHP's ftp extension), HTTP form based upload, proxies, cookies and user+password authentication.

In order to use the CURL functions you need to install the CURL (<http://curl.haxx.se/>) package. PHP requires that you use CURL 7.0.2-beta or higher. PHP will not work with any version of CURL below version 7.0.2-beta.

To use PHP's CURL support you must also compile PHP `-with-curl[=DIR]` where DIR is the location of the directory containing the lib and include directories. In the "include" directory there should be a folder named "curl" which should contain the easy.h and curl.h files. There should be a file named "libcurl.a" located in the "lib" directory.

These functions have been added in PHP 4.0.2.

Once you've compiled PHP with CURL support, you can begin using the curl functions. The basic idea behind the CURL functions is that you initialize a CURL session using the `curl_init()`, then you can set all your options for the transfer via the `curl_exec()` and then you finish off your session using the `curl_close()`. Here is an example that uses the CURL functions to fetch the PHP homepage into a file:

Example 1. Using PHP's CURL module to fetch the PHP homepage

```
<?php

$ch = curl_init ("http://www.php.net/");
$fp = fopen ("php_homepage.txt", "w");

curl_setopt ($ch, CURLOPT_FILE, $fp);
curl_setopt ($ch, CURLOPT_HEADER, 0);

curl_exec ($ch);
curl_close ($ch);
fclose ($fp);
?>
```


curl_init (PHP 4 >= 4.0.2)

Initialize a CURL session

```
int curl_init ([string url])
```

The **curl_init()** will initialize a new session and return a CURL handle for use with the **curl_setopt()**, **curl_exec()**, and **curl_close()** functions. If the optional *url* parameter is supplied then the CURLOPT_URL option will be set to the value of the parameter. You can manually set this using the **curl_setopt()** function.

Example 1. Initializing a new CURL session and fetching a webpage

```
<?php
$ch = curl_init();

curl_setopt ($ch, CURLOPT_URL, "http://www.zend.com/");
curl_setopt ($ch, CURLOPT_HEADER, 0);

curl_exec ($ch);

curl_close ($ch);
?>
```

See also: **curl_close()**, **curl_setopt()**

curl_setopt (PHP 4 >= 4.0.2)

Set an option for a CURL transfer

```
bool curl_setopt (int ch, string option, mixed value)
```

The **curl_setopt()** function will set options for a CURL session identified by the *ch* parameter. The *option* parameter is the option you want to set, and the *value* is the value of the option given by the *option*.

The *value* should be a long for the following options (specified in the *option* parameter):

- *CURLOPT_INFILESIZE*: When you are uploading a file to a remote site, this option should be used to tell PHP what the expected size of the infile will be.
- *CURLOPT_VERBOSE*: Set this option to a non-zero value if you want CURL to report everything that is happening.
- *CURLOPT_HEADER*: Set this option to a non-zero value if you want the header to be included in the output.
 - *CURLOPT_NOPROGRESS*: Set this option to a non-zero value if you don't want PHP to display a progress meter for CURL transfers

Note: PHP automatically sets this option to a non-zero parameter, this should only be changed for debugging purposes.

- *CURLOPT_NOBODY*: Set this option to a non-zero value if you don't want the body included with the output.
- *CURLOPT_FAILONERROR*: Set this option to a non-zero value if you want PHP to fail silently if the HTTP code returned is greater than 300. The default behaviour is to return the page normally, ignoring the code.
- *CURLOPT_UPLOAD*: Set this option to a non-zero value if you want PHP to prepare for an upload.

- *CURLOPT_POST*: Set this option to a non-zero value if you want PHP to do a regular HTTP POST. This POST is a normal application/x-www-form-urlencoded kind, most commonly used by HTML forms.
- *CURLOPT_FTPLISTONLY*: Set this option to a non-zero value and PHP will just list the names of an FTP directory.
- *CURLOPT_FTPAPPEND*: Set this option to a non-zero value and PHP will append to the remote file instead of overwriting it.
- *CURLOPT_NETRC*: Set this option to a non-zero value and PHP will scan your `~/.netrc` file to find your username and password for the remote site that you're establishing a connection with.
- *CURLOPT_FOLLOWLOCATION*: Set this option to a non-zero value to follow any "Location: " header that the server sends as a part of the HTTP header (note this is recursive, PHP will follow as many "Location: " headers that it is sent.)
- *CURLOPT_PUT*: Set this option a non-zero value to HTTP PUT a file. The file to PUT must be set with the *CURLOPT_INFILE* and *CURLOPT_INFILESIZE*.
- *CURLOPT_MUTE*: Set this option to a non-zero value and PHP will be completely silent with regards to the CURL functions.
- *CURLOPT_TIMEOUT*: Pass a long as a parameter that contains the maximum time, in seconds, that you'll allow the curl functions to take.
- *CURLOPT_LOW_SPEED_LIMIT*: Pass a long as a parameter that contains the transfer speed in bytes per second that the transfer should be below during *CURLOPT_LOW_SPEED_TIME* seconds for PHP to consider it too slow and abort.
- *CURLOPT_LOW_SPEED_TIME*: Pass a long as a parameter that contains the time in seconds that the transfer should be below the *CURLOPT_LOW_SPEED_LIMIT* for PHP to consider it too slow and abort.
- *CURLOPT_RESUME_FROM*: Pass a long as a parameter that contains the offset, in bytes, that you want the transfer to start from.
- *CURLOPT_SSLVERSION*: Pass a long as a parameter that contains the SSL version (2 or 3) to use. By default PHP will try and determine this by itself, although, in some cases you must set this manually.
- *CURLOPT_TIMECONDITION*: Pass a long as a parameter that defines how the *CURLOPT_TIMEVALUE* is treated. You can set this parameter to *TIMECOND_IFMODSINCE* or *TIMECOND_ISUNMODSINCE*. This is a HTTP-only feature.
- *CURLOPT_TIMEVALUE*: Pass a long as a parameter that is the time in seconds since January 1st, 1970. The time will be used as specified by the *CURLOPT_TIMEVALUE* option, or by default the *TIMECOND_IFMODSINCE* will be used.

The *value* parameter should be a string for the following values of the *option* parameter:

- *CURLOPT_URL*: This is the URL that you want PHP to fetch. You can also set this option when initializing a session with the *curl_init()* function.
- *CURLOPT_USERPWD*: Pass a string formatted in the [username]:[password] manner, for PHP to use for the connection.
- *CURLOPT_PROXYUSERPWD*: Pass a string formatted in the [username]:[password] format for connection to the HTTP proxy.
- *CURLOPT_RANGE*: Pass the specified range you want. It should be in the "X-Y" format, where X or Y may be left out. The HTTP transfers also support several intervals, separated with commas as in X-Y,N-M.
- *CURLOPT_POSTFIELDS*: Pass a string containing the full data to post in an HTTP "POST" operation.
- *CURLOPT_REFERER*: Pass a string containing the "referer" header to be used in an HTTP request.
- *CURLOPT_USERAGENT*: Pass a string containing the "user-agent" header to be used in an HTTP request.
- *CURLOPT_FTPPORT*: Pass a string containing the which will be used to get the IP address to use for the ftp "POST" instruction. The POST instruction tells the remote server to connect to our specified IP address. The string may be a plain IP address, a hostname, a network interface name (under UNIX), or just a plain '-' to use the systems default IP address.

- *CURLOPT_COOKIE*: Pass a string containing the content of the cookie to be set in the HTTP header.
- *CURLOPT_SSLCERT*: Pass a string containing the filename of PEM formatted certificate.
- *CURLOPT_SSLCERTPASSWD*: Pass a string containing the password required to use the CURLOPT_SSLCERT certificate.
- *CURLOPT_COOKIEFILE*: Pass a string containing the name of the file containing the cookiee data. The cookie file can be in Netscape format, or just plain HTTP-style headers dumped into a file.
 - *CURLOPT_CUSTOMREQUEST*: Pass a string to be used instead of GET or HEAD when doing an HTTP request. This is useful for doing DELETE or another, more obscure, HTTP request.

Note: Don't do this without making sure your server supports the command first.

The following options expect a file descriptor that is obtained by using the **fopen()** function:

- *CURLOPT_FILE*: The file where the output of your transfer should be placed, the default is STDOUT.
- *CURLOPT_INFILE*: The file where the input of your transfer comes from.
- *CURLOPT_WRITEHEADER*: The file to write the header part of the output into.
- *CURLOPT_STDERR*: The file to write errors to instead of stderr.

curl_exec (PHP 4 >= 4.0.2)

Perform a CURL session

```
bool curl_exec (int ch)
```

This function is should be called after you initialize a CURL session and all the options for the session are set. Its purpose is simply to execute the predefined CURL session (given by the *ch*).

curl_close (PHP 4 >= 4.0.2)

Close a CURL session

```
void curl_close (int ch)
```

This functions closes a CURL session and frees all ressources. The CURL handle, *ch*, is also deleted.

curl_version (PHP 4 >= 4.0.2)

Return the current CURL version

```
string curl_version (void);
```

The **curl_version()** function returns a string containing the current CURL version.

XI. Cybershell payment functions

These functions are only available if the interpreter has been compiled with the `-with-cybercash=[DIR]`. These functions have been added in PHP 4.

cybercash_encr (PHP 4 >= 4.0b4)

???

```
array cybercash_encr (string wmk, string sk, string inbuff)
```

The function returns an associative array with the elements "errcode" and, if "errcode" is false, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_decr (PHP 4 >= 4.0b4)

???

```
array cybercash_decr (string wmk, string sk, string inbuff)
```

The function returns an associative array with the elements "errcode" and, if "errcode" is false, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_base64_encode (PHP 4 >= 4.0b4)

???

```
string cybercash_base64_encode (string inbuff)
```

cybercash_base64_decode (PHP 4 >= 4.0b4)

```
string cybercash_base64_decode (string inbuff)
```


XII. 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 (<http://www.sleepycat.com/>) 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.

To add support for any of the following handlers, add the specified `--with` configure switch to your PHP configure line:

- 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. (`--with-dbm`)
- Ndbm is a newer type and more flexible than dbm. It still has most of the arbitrary limits of dbm (therefore it is deprecated). (`--with-ndbm`)
- Gdbm is the GNU database manager (<ftp://ftp.gnu.org/pub/gnu/gdbm/>). (`--with-gdbm`)
- DB2 is Sleepycat Software's DB2 (<http://www.sleepycat.com/>). It is described as "a programmatic toolkit that provides high-performance built-in database support for both standalone and client/server applications." (`--with-db2`)
- DB3 is Sleepycat Software's DB3 (<http://www.sleepycat.com/>). (`--with-db3`)
- 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 (<http://pobox.com/~djb/cdb.html>). Since it is constant, we support only reading operations. (`--with-cdb`)

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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Close database

```
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()** and **dba_popen()**

dba_delete (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Delete entry specified by key

```
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()**, and **dba_replace()**.

dba_exists (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Check whether key exists

```
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()**, and **dba_replace()**.

dba_fetch (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Fetch data specified by key

```
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()**, and **dba_replace()**.

dba_firstkey (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Fetch first key

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Insert entry

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Fetch next key

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Open database persistently

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Open database

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Optimize database

```
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 (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Replace or insert entry

```
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()**, and **dba_insert()**.

dba_sync (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Synchronize database

```
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()**

XIII. Date and Time functions

checkdate (PHP 3, PHP 4)

Validate a gregorian date/time

```
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 1 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 (PHP 3, PHP 4)

Format a local time/date

```
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"
- B - Swatch Internet time
- 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"
- 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"
- h - hour, 12-hour format; i.e. "01" to "12"
- H - hour, 24-hour format; i.e. "00" to "23"
- i - minutes; i.e. "00" to "59"
- I (capital i) - "1" if Daylight Savings Time, "0" otherwise.
- 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"
- M - month, textual, 3 letters; i.e. "Jan"
- n - month without leading zeros; i.e. "1" to "12"
- r - RFC 822 formatted date; i.e. "Thu, 21 Dec 2000 16:01:07 +0200"

- 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"
- T - Timezone setting of this machine; i.e. "MDT"
- 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"). The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.

Unrecognized characters in the format string will be printed as-is. The "Z" format will always return "0" when using **gdate()**.

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 **gdate()** and **mktime()**.

getdate (PHP 3, PHP 4)

Get date/time information

```
array getdate ([int timestamp])
```

Returns an associative array containing the date information of the *timestamp*, or the current local time if no timestamp is given, 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 (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Get current time

```
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 (PHP 3, PHP 4)

Format a GMT/CUT date/time

```
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()**, **mftime()**, and **gmmktime()**.

gmmktime (PHP 3, PHP 4)

Get UNIX timestamp for a GMT date

```
int gmmktime (int hour, int minute, int second, int month, int day, int year [, int is_dst])
```

Identical to **mftime()** except the passed parameters represents a GMT date.

gmstrftime (PHP 3>= 3.0.12, PHP 4 >= 4.0RC2)

Format a GMT/CUT time/date according to locale settings

```
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()**.

localtime (PHP 4 >= 4.0RC2)

Get the local time

```
array localtime ([int timestamp [, bool is_associative]])
```

The **localtime()** function returns an array identical to that of the structure returned by the C function call. The first argument to **localtime()** is the timestamp, if this is not given the current time is used. The second argument to the **localtime()** is the *is_associative*, if this is set to 0 or not supplied than the array is returned as a regular, numerically indexed array. If the argument is set to 1 then **localtime()** is an associative array containing all the different elements of the structure returned by the C function call to localtime. The names of the different keys of the associative array are as follows:

- "tm_sec" - seconds
- "tm_min" - minutes
- "tm_hour" - hour
- "tm_mday" - day of the month
- "tm_mon" - month of the year
- "tm_year" - Year, not y2k compliant
- "tm_wday" - Day of the week
- "tm_yday" - Day of the year
- "tm_isdst" - Is daylight savings time in effect

microtime (PHP 3, PHP 4)

Return current UNIX timestamp with microseconds

```
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()**.

mktme (PHP 3, PHP 4)

Get UNIX timestamp for a date

```
int mktme (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.

Mktme() 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. Mktme() example

```
echo date ("M-d-Y", mktme (0,0,0,12,32,1997));
echo date ("M-d-Y", mktme (0,0,0,13,1,1997));
echo date ("M-d-Y", mktme (0,0,0,1,1,1998));
echo date ("M-d-Y", mktme (0,0,0,1,1,98));
```

Year may be a two or four digit value, with values between 0-69 mapping to 2000-2069 and 70-99 to 1970-1999 (on systems where time_t is a 32bit signed integer, as most common today, the valid range for *year* is somewhere between 1902 and 2037).

The last day of any given month can be expressed as the "0" day of the next month, not the -1 day. Both of the following examples will produce the string "The last day in Feb 2000 is: 29".

Example 2. Last day of next month

```
$lastday = mktme (0,0,0,3,0,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);

$lastday = mktme (0,0,0,4,-31,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);
```

Date with year, month and day equal to zero is considered illegal (otherwise it would be regarded as 30.11.1999, which would be strange behaviour).

See also **date()** and **time()**.

strftime (PHP 3, PHP 4)

Format a local time/date according to locale settings

```
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
- %C - century number (the year divided by 100 and truncated to an integer, range 00 to 99)
- %d - day of the month as a decimal number (range 01 to 31)
- %D - same as %m/%d/%y
- %e - day of the month as a decimal number, a single digit is preceded by a space (range ' 1' to '31')
- %h - same as %b
- %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 01 to 12)
- %M - minute as a decimal number
- %n - newline character
- %p - either 'am' or 'pm' according to the given time value, or the corresponding strings for the current locale
- %r - time in a.m. and p.m. notation
- %R - time in 24 hour notation
- %S - second as a decimal number
- %t - tab character
- %T - current time, equal to %H:%M:%S
- %u - weekday as a decimal number [1,7], with 1 representing Monday
- %U - week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week
- %V - The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the 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

Note: Not all conversion specifiers may be supported by your C library, in which case they will not be supported by PHP's **strftime()**.

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()** and the Open Group specification of **strftime()** (<http://www.opengroup.org/onlinepubs/7908799/xsh/strftime.html>).

time (PHP 3, PHP 4)

Return current UNIX timestamp

```
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()**.

strtotime (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Parse about any english textual datetime description into a UNIX timestamp

```
int strtotime (string time [, int now])
```

The function expects to be given a string containing an english date format and will try to parse that format into a UNIX timestamp.

Example 1. Strtotime() examples

```
echo strtotime ("now") . "\n";
echo strtotime ("10 September 2000") . "\n";
```

```
echo strtotime ("+1 day") . "\n";
echo strtotime ("+1 week") . "\n";
echo strtotime ("+1 week 2 days 4 hours 2 seconds") . "\n";
```

XIV. 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 (PHP 3, PHP 4)

Creates a dBase database

```
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 (PHP 3, PHP 4)

Opens a dBase database

```
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 (PHP 3, PHP 4)

Close a dBase database

```
bool dbase_close (int dbase_identifier)
```

Closes the database associated with *dbase_identifier*.

dbase_pack (PHP 3, PHP 4)

Packs a dBase database

```
bool dbase_pack (int dbase_identifier)
```

Packs the specified database (permanently deleting all records marked for deletion using **dbase_delete_record()**).

dbase_add_record (PHP 3, PHP 4)

Add a record to a dBase database

```
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 (PHP 3>= 3.0.11, PHP 4)

Replace a record in a dBase database

```
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 (PHP 3, PHP 4)

Deletes a record from a dBase database

```
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 (PHP 3, PHP 4)

Gets a record from a dBase database

```
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 (PHP 3>= 3.0.4, PHP 4)

Gets a record from a dBase database as an associative array

```
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 (PHP 3, PHP 4)

Find out how many fields are in a dBase database

```
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 (PHP 3, PHP 4)

Find out how many records are in a dBase database

```
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.

XV. 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 (PHP 3, PHP 4)

Opens a DBM database

```
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 ([ftp://ftp.gnu.org/pub/gnu/gdbm/](http://ftp.gnu.org/pub/gnu/gdbm/)).

dbmclose (PHP 3, PHP 4)

Closes a dbm database

```
bool dbmclose (int dbm_identifier)
```

Unlocks and closes the specified database.

dbmexists (PHP 3, PHP 4)

Tells if a value exists for a key in a DBM database

```
bool dbmexists (int dbm_identifier, string key)
```

Returns TRUE if there is a value associated with the *key*.

dbmfetch (PHP 3, PHP 4)

Fetches a value for a key from a DBM database

```
string dbmfetch (int dbm_identifier, string key)
```

Returns the value associated with *key*.

dbminsert (PHP 3, PHP 4)

Inserts a value for a key in a DBM database

```
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 (PHP 3, PHP 4)

Replaces the value for a key in a DBM database

```
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 (PHP 3, PHP 4)

Deletes the value for a key from a DBM database

```
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.

dbmfirkey (PHP 3, PHP 4)

Retrieves the first key from a DBM database

```
string dbmfirkey (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.

dbmnxtkey (PHP 3, PHP 4)

Retrieves the next key from a DBM database

```
string dbmnxtkey (int dbm_identifier, string key)
```

Returns the next key after *key*. By calling **dbmfirkey()** followed by successive calls to **dbmnxtkey()** 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

```
$key = dbmfirkey ($dbm_id);
while ($key) {
    echo "$key = " . dbmfetch ($dbm_id, $key) . "\n";
```

```
    $key = dbmnextkey ($dbm_id, $key);  
}
```

dblist (PHP 3, PHP 4)

Describes the DBM-compatible library being used

```
string dblist (void)
```


XVI. Directory functions

chdir (PHP 3, PHP 4)

change directory

```
int chdir (string directory)
```

Changes PHP's current directory to *directory*. Returns FALSE if unable to change directory, TRUE otherwise.

dir (PHP 3, PHP 4)

directory class

```
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 (PHP 3, PHP 4)

close directory handle

```
void closedir (int dir_handle)
```

Closes the directory stream indicated by *dir_handle*. The stream must have previously been opened by **opendir()**.

getcwd (PHP 4 >= 4.0b4)

gets the current working directory

```
string getcwd(void);
```

Returns the current working directory.

opendir (PHP 3, PHP 4)

open directory handle

```
int opendir (string path)
```

Returns a directory handle to be used in subsequent **closedir()**, **readdir()**, and **rewinddir()** calls.

readdir (PHP 3, PHP 4)

read entry from directory handle

```
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

```
// Note that !== did not exist until 4.0.0-RC2
<?php
$handle=opendir('.');
echo "Directory handle: $handle\n";
echo "Files:\n";
while (($file = readdir($handle))!==false) {
    echo "$file\n";
}
closedir($handle);
?>
```

Note that **readdir()** will return the . and .. entries. If you don't want these, simply strip them out:

Example 2. List all files in the current directory and strip out . and ..

```
<?php
$handle=opendir('.');
while (false!==($file = readdir($handle))) {
    if ($file != "." && $file != "..") {
        echo "$file\n";
    }
}
closedir($handle);
?>
```

rewinddir (PHP 3, PHP 4)

rewind directory handle

```
void rewinddir (int dir_handle)
```

Resets the directory stream indicated by *dir_handle* to the beginning of the directory.

XVII. DOM XML functions

This documentation is not finished yet. Don't start to translate it or use it as a programming reference (steinm@php.net).

These functions are only available if PHP was configured with `-with-dom=[DIR]`, using the GNOME xml library. You will need at least libxml-2.2.7 These functions have been added in PHP 4.

The extension allows to operate on an XML document with the DOM api. It also provides a function `xmmtree()` to turn the complete XML document into a tree of php objects. Currently this tree is read only, which doesn't mean you cannot modify it, but it would make any sense since `dumpmem()` cannot be applied to it. Therefore, if you want to read an XML file and write a modified version use the `add_node()`, `set_attribute()`, etc. and finally `dumpmem()` functions.

This module defines the following constants:

Table 1. XML constants

Constant	Value	Description
XML_ELEMENT_NODE	1	Node is an element
XML_ATTRIBUTE_NODE	2	Node is an attribute
XML_TEXT_NODE	3	Node is a piece of text
XML_CDATA_SECTION_NODE	4	
XML_ENTITY_REF_NODE	5	
XML_ENTITY_NODE	6	Node is an entity like
XML_PI_NODE	7	Node is a processing instruction
XML_COMMENT_NODE	8	Node is a comment
XML_DOCUMENT_NODE	9	Node is a document
XML_DOCUMENT_TYPE_NODE	10	
XML_DOCUMENT_FRAG_NODE	11	
XML_NOTATION_NODE	12	
XML_GLOBAL_NAMESPACE	1	
XML_LOCAL_NAMESPACE	2	

This module defines a number of classes, which are listed — including their properties and method — in the following table.

Table 2. DomDocument class (methods)

Name	PHP name	Description
root	domxml_root	
children	domxml_children	
add_root	domxml_add_root	
dtd	domxml_intdtd	
dumpmem	domxml	
xpath_init	xpath_init	
xpath_new_context	xpath_new_context	
xptr_new_context	xptr_new_context	

Table 3. DomDocument class (attributes)

Name	Type	Description
doc	class DomDocument	The object itself
name	string	
url	string	
version	string	Version of XML
encoding	string	
standalone	long	1 if the file is a standalone version
type	long	One of the constants in table ...
compression	long	1 if the file is compressed
charset	long	

Table 4. DomNode class (methods)

Name	PHP name	Description
lastchild	domxml_last_child()	
children	domxml_children()	
parent	domxml_parent()	
new_child	domxml_new_child()	
get_attribute	domxml_get_attribute()	
set_attribute	domxml_set_attribute()	
attributes	domxml_attributes()	
node	domxml_node()	
set_content()	domxml_set_content	

Table 5. DomNode class (attributes)

Name	Type	Description
node	class DomNode	The object itself
type	long	
name	string	
content	string	

xmlDoc (PHP 4 >= 4.0b4)

Creates a DOM object of an XML document

```
object xmlDoc (string str)
```

The function parses the XML document in *str* and returns an object of class "Dom document", having the properties as listed above.

See also **xmlDocfile()**

xmlDocfile (PHP 4 >= 4.0b4)

Creates a DOM object from XML file

```
object xmlDocfile (string filename)
```

The function parses the XML document in the file named *filename* and returns an object of class "Dom document", having the properties as listed above. The file is accessed read only.

See also **xmlDoc()**

xmlTree (PHP 4 >= 4.0b4)

Creates a tree of php objects from XML document

```
object xmlTree (string str)
```

The function parses the XML document in *str* and returns a tree PHP objects as the parsed document. This function is isolated from the other functions, which means, you cannot access the tree with any of the other functions. Modifying it, for example by adding nodes, makes no sense since there is currently no way to dump it as an XML file. Though this function may be valuable if you want to read a file and investigate the content.

domxml_root (PHP 4 >= 4.0b4)

Returns array of root nodes

```
object domxml_root (array doc)
```

Returns an array of nodes located at the root of a an DOM document.

domxml_add_root (PHP 4 >= 4.0b4)

Adds a further root node

```
resource domxml_add_root (resource doc)
```

Adds a root node to a dom document and returns the new node.

domxml_dumpmem (PHP 4 >= 4.0b4)

Dumps the internal XML tree back into a string

```
string domxml_dumpmem (resource doc)
```

Creates an XML document from the dom representation.

domxml_attributes (PHP 4 >= 4.0b4)

Returns an array of attributes of a node

```
array domxml_attributes (resource node)
```

Returns all attributes of a node as array of objects of type "dom attribute".

domxml_get_attribute (unknown)

Returns a certain attribute of a node

```
object domxml_get_attribute (resource node, string name)
```

Returns the attribute with name *name* of the given node.

domxml_set_attribute (unknown)

```
object domxml_set_attribute (resource node, string name, string value)
```

Sets an attribute with name *name* of the given node on a value.

domxml_children (PHP 4 >= 4.0b4)

Returns children of a node or document

```
array domxml_children (object doc/node)
```

Returns all children of a node as an array of nodes.

domxml_new_child (PHP 4 >= 4.0b4)

Adds new child node

```
resource domxml_new_child (resource node)
```

Adds a new child to a node and returns it.

domxml_new_xmldoc (PHP 4 >= 4.0b4)

Creates new empty XML document

```
object domxml_new_xmldoc (string version)
```

Creates a new dom document from scratch and returns it.

domxml_node (PHP 4 >= 4.0b4)

Creates new node

```
resource domxml_node (string name)
```

Create a new node.

XVIII. Error Handling and Logging Functions

These are functions dealing with error handling and logging. They allow you to define your own error handling rules, as well as modify the way the errors can be logged. This allows you to change and enhance error reporting to suit your needs.

With the logging functions, you can send messages directly to other machines, to an email (or email to pager gateway!), to system logs, etc., so you can selectively log and monitor the most important parts of your applications and websites.

The error reporting functions allow you to customize what level and kind of error feedback is given, ranging from simple notices to customized functions returned during errors.

error_log (PHP 3, PHP 4)

send an error message somewhere

```
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</i> 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.
1	<i>message</i> is sent by email to the address in the <i>destination</i> parameter. This is the only message type where the fourth parameter, <i>extra_headers</i> is used. This message type uses the same internal function as Mail() does.
2	<i>message</i> is sent through the PHP debugging connection. This option is only available if remote debugging has been enabled . In this case, the <i>destination</i> parameter specifies the host name or IP address and optionally, port number, of the socket receiving the debug information.
3	<i>message</i> is appended to the file <i>destination</i> .

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 (PHP 3, PHP 4)

set which PHP errors are reported

```
int error_reporting ([int level])
```

Sets PHP's error reporting level and returns the old level. The error reporting level is either a bitmask, or named constant. Using named constants is strongly encouraged to ensure compatibility for future versions. As error levels are added, the range of integers increases, so older integer-based error levels will not always behave as expected.

Example 1. Error Integer changes

```
error_reporting (55); // PHP 3 equivalent to E_ALL ^ E_NOTICE

/* ...in PHP 4, '55' would mean (E_ERROR | E_WARNING | E_PARSE |
E_CORE_ERROR | E_CORE_WARNING) */

error_reporting (2039); // PHP 4 equivalent to E_ALL ^ E_NOTICE

error_reporting (E_ALL ^ E_NOTICE); // The same in both PHP 3 and 4
```

Follow the links for the internal values to get their meanings:

Table 1. error_reporting() bit values

constant	value
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING
64	E_COMPILE_ERROR
128	E_COMPILE_WARNING
256	E_USER_ERROR
512	E_USER_WARNING
1024	E_USER_NOTICE

Example 2. error_reporting() examples

```
error_reporting(0);
/* Turn off all reporting */

error_reporting (7); // Old syntax, PHP 2/3
error_reporting (E_ERROR | E_WARNING | E_PARSE); // New syntax for PHP 3/4
/* Good to use for simple running errors */

error_reporting (15); // Old syntax, PHP 2/3
error_reporting (E_ERROR | E_WARNING | E_PARSE | E_NOTICE); // New syntax for PHP 3/4
/* good for code authoring to report uninitialized or (possibly mis-spelled) variables */

error_reporting (63); // Old syntax, PHP 2/3
error_reporting (E_ALL); // New syntax for PHP 3/4
```

```
/* report all PHP errors */
```

restore_error_handler (PHP 4 >= 4.0.1)

Restores the previous error handler function

```
void restore_error_handler (void)
```

Used after changing the error handler function using **set_error_handler()**, to revert to the previous error handler (which could be the built-in or a user defined function)

See also **error_reporting()**, **set_error_handler()**, **trigger_error()**, **user_error()**

set_error_handler (PHP 4 >= 4.0.1)

Sets a user-defined error handler function.

```
string set_error_handler (string error_handler)
```

Sets a user function (*error_handler*) to handle errors in a script. Returns the previously defined error handler (if any), or false on error. This function can be used for defining your own way of handling errors during runtime, for example in applications in which you need to do cleanup of data/files when a critical error happens, or when you need to trigger an error under certain conditions (using **trigger_error()**)

The user function needs to accept 2 parameters: the error code, and a string describing the error. From PHP 4.0.2, an additional 3 optional parameters are supplied: the filename in which the error occurred, the line number in which the error occurred, and the context in which the error occurred (an array that points to the active symbol table at the point the error occurred).

The example below shows the handling of internal exceptions by triggering errors and handling them with a user defined function:

Example 1. Error handling with set_error_handler() and trigger_error()

```
<?php

// redefine the user error constants - PHP 4 only
define (FATAL,E_USER_ERROR);
define (ERROR,E_USER_WARNING);
define (WARNING,E_USER_NOTICE);

// set the error reporting level for this script
error_reporting (FATAL | ERROR | WARNING);

// error handler function
function myErrorHandler ($errno, $errstr, $errfile, $errline) {
    switch ($errno) {
        case FATAL:
            echo "<b>FATAL</b> [$errno] $errstr<br>\n";
            echo " Fatal error in line ".$errline." of file ".$errfile;
            echo ", PHP ".PHP_VERSION." (".PHP_OS.")<br>\n";
            echo "Aborting...<br>\n";
            exit -1;
    }
}
```

```

        break;
    case ERROR:
        echo "<b>ERROR</b> [{$errno}] $errstr<br>\n";
        break;
    case WARNING:
        echo "<b>WARNING</b> [{$errno}] $errstr<br>\n";
        break;
    default:
        echo "Unknown error type: [{$errno}] $errstr<br>\n";
        break;
    }
}

// function to test the error handling
function scale_by_log ($vect, $scale) {
    if ( !is_numeric($scale) || $scale <= 0 )
        trigger_error("log(x) for x <= 0 is undefined, you used: scale = $scale",
                      FATAL);
    if (!is_array($vect)) {
        trigger_error("Incorrect input vector, array of values expected", ERROR);
        return null;
    }
    for ($i=0; $i<count($vect); $i++) {
        if (!is_numeric($vect[$i]))
            trigger_error("Value at position $i is not a number, using 0 (zero)",
                          WARNING);
        $temp[$i] = log($scale) * $vect[$i];
    }
    return $temp;
}

// set to the user defined error handler
$old_error_handler = set_error_handler("myErrorHandler");

// trigger some errors, first define a mixed array with a non-numeric item
echo "vector a\n";
$a = array(2,3,"foo",5.5,43.3,21.11);
print_r($a);

// now generate second array, generating a warning
echo "---\nvector b - a warning (b = log(PI) * a)\n";
$b = scale_by_log($a, M_PI);
print_r($b);

// this is trouble, we pass a string instead of an array
echo "---\nvector c - an error\n";
$c = scale_by_log("not array",2.3);
var_dump($c);

// this is a critical error, log of zero or negative number is undefined
echo "---\nvector d - fatal error\n";
$d = scale_by_log($a, -2.5);

?>

```

And when you run this sample script, the output will be

```

vector a
Array
(
    [0] => 2
)

```

```

[1] => 3
[2] => foo
[3] => 5.5
[4] => 43.3
[5] => 21.11
)
---
vector b - a warning (b = log(PI) * a)
<b>WARNING</b> [1024] Value at position 2 is not a number, using 0 (zero)<br>
Array
(
    [0] => 2.2894597716988
    [1] => 3.4341896575482
    [2] => 0
    [3] => 6.2960143721717
    [4] => 49.566804057279
    [5] => 24.165247890281
)
---
vector c - an error
<b>ERROR</b> [512] Incorrect input vector, array of values expected<br>
NULL
---
vector d - fatal error
<b>FATAL</b> [256] log(x) for x <= 0 is undefined, you used: scale = -2.5<br>
    Fatal error in line 36 of file trigger_error.php, PHP 4.0.2 (Linux)<br>
Aborting...<br>
```

It is important to remember that the standard PHP error handler is completely bypassed. **error_reporting()** settings will have no effect and your error handler will be called regardless - however you are still able to read the current value of **error_reporting()** and act appropriately. Of particular note is that this value will be 0 if the statement that caused the error was prepended by the [@ error-control operator](#).

Also note that it is your responsibility to **die()** if necessary. If the error-handler function returns, script execution will continue with the next statement after the one that caused an error.

See also **error_reporting()**, **restore_error_handler()**, **trigger_error()**, **user_error()**

trigger_error (PHP 4 >= 4.0.1)

Generates a user-level error/warning/notice message

```
void trigger_error (string error_msg [, int error_type])
```

Used to trigger a user error condition, it can be used by in conjunction with the built-in error handler, or with a user defined function that has been set as the new error handler (**set_error_handler()**). It only works with the E_USER family of constants, and will default to E_USER_NOTICE.

This function is useful when you need to generate a particular response to an exception at runtime. For example:

```
if (assert ($divisor == 0))
    trigger_error ("Cannot divide by zero", E_USER_ERROR);
```

Note: See **set_error_handler()** for a more extensive example.

See also **error_reporting()**, **set_error_handler()**, **restore_error_handler()**, **user_error()**

user_error (PHP 4 >= 4.0RC2)

Generates a user-level error/warning/notice message

```
void user_error (string error_msg [, int error_type])
```

This is an alias for the function **trigger_error()**.

See also **error_reporting()**, **set_error_handler()**, **restore_error_handler()**, and **trigger_error()**

XIX. 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 (PHP 3, PHP 4)

read and verify the map file

```
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 (PHP 3, PHP 4)

gets the name of a field

```
string filepro_fieldname (int field_number)
```

Returns the name of the field corresponding to *field_number*.

filepro_fieldtype (PHP 3, PHP 4)

gets the type of a field

```
string filepro_fieldtype (int field_number)
```

Returns the edit type of the field corresponding to *field_number*.

filepro_fieldwidth (PHP 3, PHP 4)

gets the width of a field

```
int filepro_fieldwidth (int field_number)
```

Returns the width of the field corresponding to *field_number*.

filepro_retrieve (PHP 3, PHP 4)

retrieves data from a filePro database

```
string filepro_retrieve (int row_number, int field_number)
```

Returns the data from the specified location in the database.

filepro_fieldcount (PHP 3, PHP 4)

find out how many fields are in a filePro database

```
int filepro_fieldcount(void);
```

Returns the number of fields (columns) in the opened filePro database.

See also [filepro\(\)](#).

filepro_rowcount (PHP 3, PHP 4)

find out how many rows are in a filePro database

```
int filepro_rowcount(void);
```

Returns the number of rows in the opened filePro database.

See also [filepro\(\)](#).

XX. Filesystem functions

basename (PHP 3, PHP 4)

Returns filename component of path

```
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 (PHP 3, PHP 4)

Changes file group

```
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.

See also [chown\(\)](#) and [chmod\(\)](#).

Note: This function does not work on Windows systems

chmod (PHP 3, PHP 4)

Changes file mode

```
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, so strings (such as "g+w") will not work properly. To ensure the expected operation, you need to prefix *mode* with a zero (0):

```
chmod ("somedir/somefile", 755);    // decimal; probably incorrect
chmod ("somedir/somefile", "u+rwx,go+r"); // string; incorrect
chmod ("somedir/somefile", 0755);    // octal; correct value of mode
```

Returns true on success and false otherwise.

See also **chown()** and **chgrp()**.

Note: This function does not work on Windows systems

chown (PHP 3, PHP 4)

Changes file owner

```
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.

See also **chown()** and **chmod()**.

Note: This function does not work on Windows systems

clearstatcache (PHP 3, PHP 4)

Clears file stat cache

```
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_writable()**, **is_readable()**, **is_executable()**, **is_file()**, **is_dir()**, **is_link()**, **filectime()**, **fileatime()**, **filemtime()**, **fileinode()**, **filegroup()**, **fileowner()**, **filesize()**, **filetype()**, and **fileperms()**.

copy (PHP 3, PHP 4)

Copies file

```
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 (unknown)

A dummy manual entry

```
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 (PHP 3, PHP 4)

Returns directory name component of path

```
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 (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Returns available space in directory

```
float diskfreespace (string directory)
```

Given a string containing a directory, this function will return the number of bytes available on the corresponding filesystem or disk partition.

Example 1. diskfreespace() example

```
$df = diskfreespace( "/"); // $df contains the number of bytes
                           // available on "/"
```

fclose (PHP 3, PHP 4)

Closes an open file pointer

```
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 **fsckopen()**.

feof (PHP 3, PHP 4)

Tests for end-of-file on a file pointer

```
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 **fsckopen()**.

fflush (PHP 4 >= 4.0.1)

Flushes the output to a file

```
int fflush (int fp)
```

This function forces a write of all buffered output to the resource pointed to by the file handle *fp*. Returns true if successful, false otherwise.

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsckopen()**.

fgetc (PHP 3, PHP 4)

Gets character from file pointer

```
string fgetc (int fp)
```

Returns a string containing a single character read from the file pointed to by *fp*. Returns FALSE on EOF.

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsckopen()**.

See also **fread()**, **fopen()**, **popen()**, **fsckopen()**, and **fgets()**.

fgetcsv (PHP 3>= 3.0.8, PHP 4)

Gets line from file pointer and parse for CSV fields

```
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 **fssockopen()**

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.

N.B. A blank line in a CSV file will be returned as an array comprising a 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 (PHP 3, PHP 4)

Gets line from file pointer

```
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 **fssockopen()**.

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()**, **fssockopen()**, and **socket_set_timeout()**.

fgetss (PHP 3, PHP 4)

Gets line from file pointer and strip HTML tags

```
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 (PHP 3, PHP 4)

Reads entire file into an array

```
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.

```
<?php
// get a web page into an array and print it out
$contents = file ('http://www.php.net');
while (list ($line_num, $line) = each ($contents)) {
    echo "<b>Line $line_num:</b> " . htmlspecialchars ($line) . "<br>\n";
}

// get a web page into a string
$contents = join ("", file ('http://www.php.net'));
?>
```

See also **readfile()**, **fopen()**, **fsockopen()**, and **popen()**.

file_exists (PHP 3, PHP 4)

Checks whether a file exists

```
bool file_exists (string filename)
```

Returns true if the file specified by *filename* exists; false otherwise.

file_exists() will not work on remote files; the file to be examined must be accessible via the server's filesystem.

The results of this function are cached. See **clearstatcache()** for more details.

fileatime (PHP 3, PHP 4)

Gets last access time of file

```
int fileatime (string filename)
```

Returns the time the file was last accessed, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

Note: The atime of a file is supposed to change whenever the data blocks of a file are being read. This can be costly performancewise when an application regularly accesses a very large number of files or directories. Some Unix filesystems can be mounted with atime updates disabled to increase the performance of such applications; USENET news spools are a common example. On such filesystems this function will be useless.

filectime (PHP 3, PHP 4)

Gets inode change time of file

```
int filectime (string filename)
```

Returns the time the file was last changed, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

Note: In most Unix filesystem, a file is considered changed, when it's Inode data is changed, that is, when the permissions, the owner, the group or other metadata from the Inode is written to. See also **filemtime()** (this is what you want to use when you want to create "Last Modified" footers on web pages) and **fileatime()**.

Note: In some Unix texts the ctime of a file is being referred to as the creation time of the file. This is wrong. There is no creation time for Unix files in most Unix filesystems.

filegroup (PHP 3, PHP 4)

Gets file group

```
int filegroup (string filename)
```

Returns the group ID of the owner of the file, or false in case of an error. The group ID is returned in numerical format, use **posix_getgrgid()** to resolve it to a group name.

The results of this function are cached. See **clearstatcache()** for more details.

Note: This function does not work on Windows systems

fileinode (PHP 3, PHP 4)

Gets file inode

```
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.

Note: This function does not work on Windows systems

filemtime (PHP 3, PHP 4)

Gets file modification time

```
int filemtime (string filename)
```

Returns the time the file was last modified, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

Note: This function returns the time when the data blocks of a file were being written to, that is, the time when the content of the file was changed. Use **date()** on the result of this function to get a printable modification date for use in page footers.

fileowner (PHP 3, PHP 4)

Gets file owner

```
int fileowner (string filename)
```

Returns the user ID of the owner of the file, or false in case of an error. The user ID is returned in numerical format, use **posix_getpwuid()** to resolve it to a username.

The results of this function are cached. See **clearstatcache()** for more details.

Note: This function does not work on Windows systems

fileperms (PHP 3, PHP 4)

Gets file permissions

```
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 (PHP 3, PHP 4)

Gets file size

```
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 (PHP 3, PHP 4)

Gets file type

```
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 (PHP 3>= 3.0.7, PHP 4)

Portable advisory file locking

```
bool flock (int fp, int operation [, int wouldblock])
```

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 LOCK_SH (set to 1 prior to PHP 4.0.1).
- To acquire an exclusive lock (writer), set *operation* to LOCK_EX (set to 2 prior to PHP 4.0.1).
- To release a lock (shared or exclusive), set *operation* to LOCK_UN (set to 3 prior to PHP 4.0.1).
- If you don't want **flock()** to block while locking, add LOCK_NB (4 prior to PHP 4.0.1) 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). The optional 3rd argument is set to true if the lock would block (EWOULDBLOCK errno condition)

Flock() returns true on success and false on error (e.g. when a lock could not be acquired).

Warning

On most operation systems **flock()** is implemented at the process level. When using a multithreaded server API like ISAPI you cannot rely on **flock()** to protect files against other PHP scripts running in parallel threads of the same server instance!

fopen (PHP 3, PHP 4)

Opens file or URL

```
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. A 'Host:' header is sent with the request in order to handle name-based virtual hosts.

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 or writing via ftp (but not both simultaneously).

If *filename* is one of "php://stdin", "php://stdout", or "php://stderr", the corresponding stdio stream will be opened. (This was introduced in PHP 3.0.13; in earlier versions, a filename such as "/dev/stdin" or "/dev/fd/0" must be used to access the stdio streams.)

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.

The *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 ( "/home/rasmus/file.gif" , "wb" );
$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()**, **fsckopen()**, **socket_set_timeout()**, and **popen()**.

fpassthru (PHP 3, PHP 4)

Output all remaining data on a file pointer

```
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 **fsocckopen()**. 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 **fsocckopen()**

fputs (PHP 3, PHP 4)

Writes to a file pointer

```
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 (PHP 3, PHP 4)

Binary-safe file read

```
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()**, **fsocckopen()**, **popen()**, **fgets()**, **fgetss()**, **fscanf()**, **file()**, and **fpassthru()**.

fscanf (PHP 4 >= 4.0.1)

Parses input from a file according to a format

```
mixed fscanf (int handle, string format [, string var1...])
```

The function **fscanf()** is similar to **sscanf()**, but it takes its input from a file associated with *handle* and interprets the input according to the specified *format*. If only two parameters were passed to this function, the values parsed will be returned as an array. Otherwise, if optional parameters are passed, the function will return the number of assigned values. The optional parameters must be passed by reference.

Example 1. Fscanf() Example

```
$fp = fopen ("users.txt", "r");
while ($userinfo = fscanf ($fp, "%s\t%s\t%s\n")) {
    list ($name, $profession, $countrycode) = $userinfo;
    //... do something with the values
}
fclose($fp);
```

Example 2. users.txt

```
javier argonaut      pe
hiroshi sculptor     jp
robert slacker us
luigi florist it
```

See also **fread()**, **fgets()**, **fgetss()**, **sscanf()**, **printf()**, and **sprintf()**.

fseek (PHP 3, PHP 4)

Seeks on a file pointer

```
int fseek (int fp, int offset [, int whence])
```

Sets the file position indicator for the file referenced by *fp*. The new position, measured in bytes from the beginning of the file, is obtained by adding *offset* to the position specified by *whence*, whose values are defined as follows:

SEEK_SET - Set position equal to *offset bytes*.

SEEK_CUR - Set position to current location plus *offset*.

SEEK_END - Set position to end-of-file plus *offset*.

If *whence* is not specified, it is assumed to be **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.

Note: The *whence* argument was added after PHP 4.0 RC1.

See also **ftell()** and **rewind()**.

fstat (PHP 4 >= 4.0RC1)

Gets information about a file using an open file pointer

```
array fstat (int fp)
```

Gathers the statistics of the file opened by the file pointer *fp*. This function is similar to the **stat()** function except that it operates on an open file pointer instead of a filename.

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.

f.tell (PHP 3, PHP 4)

Tells file pointer read/write position

```
int f.tell (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()**.

ftruncate (PHP 4 >= 4.0RC1)

Truncates a file to a given length.

```
int ftruncate (int fp, int size)
```

Takes the filepointer, *fp*, and truncates the file to length, *size*. This function returns true on success and false on failure.

fwrite (PHP 3, PHP 4)

Binary-safe file write

```
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()**, **fsckopen()**, **popen()**, and **fputs()**.

set_file_buffer (PHP 3>= 3.0.8, PHP 4 >= 4.0.1)

Sets file buffering on the given file pointer

```
int set_file_buffer (int fp, int buffer)
```

Output using **fwrite()** is normally buffered at 8K. This means that if there are two processes wanting to write to the same output stream (a file), each is paused after 8K of data to allow the other to write. **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. This ensures that all writes with **fwrite()** are completed before other processes are allowed to write to that output stream.

The function returns 0 on success, or EOF if the request cannot be honored.

The following example demonstrates how to use **set_file_buffer()** to create an unbuffered stream.

Example 1. **set_file_buffer()** example

```
$fp=fopen($file, "w");
if($fp){
    set_file_buffer($fp, 0);
    fputs($fp, $output);
    fclose($fp);
}
```

See also **fopen()**, **fwrite()**.

is_dir (PHP 3, PHP 4)

Tells whether the filename is a directory

```
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 (PHP 3, PHP 4)

Tells whether the filename is executable

```
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 (PHP 3, PHP 4)

Tells whether the filename is a regular file

```
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 (PHP 3, PHP 4)

Tells whether the filename is a symbolic link

```
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()**.

Note: This function does not work on Windows systems

is_readable (PHP 3, PHP 4)

Tells whether the filename is readable

```
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_writable()**.

is_writable (PHP 4 >= 4.0b2)

Tells whether the filename is writable

```
bool is_writable (string filename)
```

Returns true if the filename exists and is writable. 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()**.

is_uploaded_file (PHP 3>= 3.0.17, PHP 4 >= 4.0.3)

Tells whether the file was uploaded via HTTP POST.

```
bool is_uploaded_file (string filename)
```

This function is available only in versions of PHP 3 after PHP 3.0.16, and in versions of PHP 4 after 4.0.2.

Returns true if the file named by *filename* was uploaded via HTTP POST. This is useful to help ensure that a malicious user hasn't tried to trick the script into working on files upon which it should not be working—for instance, `/etc/passwd`.

This sort of check is especially important if there is any chance that anything done with uploaded files could reveal their contents to the user, or even to other users on the same system.

See also **move_uploaded_file()**, and the section [Handling file uploads](#) for a simple usage example.

link (PHP 3, PHP 4)

Create a hard link

```
int link (string target, string link)
```

Link() creates a hard link.

See also the **symlink()** to create soft links, and **readlink()** along with **linkinfo()**.

Note: This function does not work on Windows systems

linkinfo (PHP 3, PHP 4)

Gets information about a link

```
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\(\)](#).

Note: This function does not work on Windows systems

mkdir (PHP 3, PHP 4)

Makes directory

```
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\(\)](#).

move_uploaded_file (PHP 4 >= 4.0.3)

Moves an uploaded file to a new location.

```
bool move_uploaded_file (string filename, string destination)
```

This function is available only in versions of PHP 3 after PHP 3.0.16, and in versions of PHP 4 after 4.0.2.

This function checks to ensure that the file designated by *filename* is a valid upload file (meaning that it was uploaded via PHP's HTTP POST upload mechanism). If the file is valid, it will be moved to the filename given by *destination*.

If *filename* is not a valid upload file, then no action will occur, and **move_uploaded_file()** will return `false`.

If *filename* is a valid upload file, but cannot be moved for some reason, no action will occur, and **move_uploaded_file()** will return `false`. Additionally, a warning will be issued.

This sort of check is especially important if there is any chance that anything done with uploaded files could reveal their contents to the user, or even to other users on the same system.

See also [is_uploaded_file\(\)](#), and the section [Handling file uploads](#) for a simple usage example.

pclose (PHP 3, PHP 4)

Closes process file pointer

```
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 (PHP 3, PHP 4)

Opens process file pointer

```
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 (PHP 3, PHP 4)

Outputs a file

```
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 **fpassthru()**, **file()**, **fopen()**, **include()**, **require()**, and **virtual()**.

readlink (PHP 3, PHP 4)

Returns the target of a symbolic link

```
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()**.

Note: This function does not work on Windows systems

rename (PHP 3, PHP 4)

Renames a file

```
int rename (string oldname, string newname)
```

Attempts to rename *oldname* to *newname*.

Returns true on success and false on failure.

rewind (PHP 3, PHP 4)

Rewind the position of a file pointer

```
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 (PHP 3, PHP 4)

Removes directory

```
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 (PHP 3, PHP 4)

Gives information about a file

```
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 (PHP 3>= 3.0.4, PHP 4)

Gives information about a file or symbolic link

```
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. 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.

realpath (PHP 4 >= 4.0b4)

Returns canonicalized absolute pathname

```
string realpath (string path)
```

realpath() expands all symbolic links and resolves references to `'./'`, `'..'` and extra `'/'` characters in the input *path* and return the canonicalized absolute pathname. The resulting path will have no symbolic link, `'./'` or `'..'` components.

Example 1. realpath() example

```
$real_path = realpath ("../../index.php");
```

symlink (PHP 3, PHP 4)

Creates a symbolic link

```
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()**.

Note: This function does not work on Windows systems.

tempnam (PHP 3, PHP 4)

Creates unique file name

```
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" );
```

Note: This function's behavior changed in 4.0.3. The temporary file is also created to avoid a race condition where the file might appear in the filesystem between the time the string was generated and before the script gets around to creating the file.

See also **tmpfile()**.

tmpfile (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Creates a temporary file

```
int tmpfile (void)
```

Creates a temporary file with an unique name in write mode, returning a file handle similar to the one returned by **fopen()**. The file is automatically removed when closed (using **fclose()**), or when the script ends.

For details, consult your system documentation on the **tmpfile(3)** function, as well as the **stdio.h** header file.

See also **tempnam()**.

touch (PHP 3, PHP 4)

Sets modification time of file

```
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.

Example 1. Touch() example

```
if (touch ($FileName)) {
    print "$FileName modification time has been
          changed to todays date and time";
} else {
    print "Sorry Could Not change modification time of $FileName";
}
```

umask (PHP 3, PHP 4)

Changes the current umask

```
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.

Note: This function may not work on Windows systems.

unlink (PHP 3, PHP 4)

Deletes a file

```
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.

Note: This function may not work on Windows systems.

XXI. 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: 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 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 MimeType application/vnd.fdf. The Acrobat reader plugin of your browser recognizes the MimeType, reads the associated PDF form and fills in the data from the FDF document.

If you look at an FDF-document with a text editor you will find a catalogue object with the name **FDF**. Such an object may contain a number of entries like **Fields**, **F**, **Status** etc.. The most commonly used entries are **Fields** which points to a list of input fields, and **F** which contains the filename of the PDF-document this data belongs to. Those entries are referred to in the FDF documentation as **/F-Key** or **/Status-Key**. Modifying this entries is done by functions like **fdf_set_file()** and **fdf_set_status()**. Fields are modified with **fdf_set_value()**, **fdf_set_opt()** etc..

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 (PHP 3>= 3.0.6, PHP 4)

Open a FDF document

```
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 (PHP 3>= 3.0.6, PHP 4)

Close an FDF document

```
boolean fdf_close (int fdf_document)
```

The **fdf_close()** function closes the FDF document.

See also **fdf_open()**.

fdf_create (PHP 3>= 3.0.6, PHP 4)

Create a new FDF document

```
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 (PHP 3>= 3.0.6, PHP 4)

Save a FDF document

```
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 (PHP 3>= 3.0.6, PHP 4)

Get the value of a field

```
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 (PHP 3>= 3.0.6, PHP 4)

Set the value of a field

```
boolean 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 (PHP 3>= 3.0.6, PHP 4)

Get the next field name

```
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 parameter is NULL.

See also **fdf_set_field()**, **fdf_get_field()**.

fdf_set_ap (PHP 3>= 3.0.6, PHP 4)

Set the appearance of a field

```
boolean fdf_set_ap (int fdf_document, string fieldname, 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=PDFNormalAP, 2=FDFRolloverAP, 3=FDFDownAP.

fdf_set_status (PHP 3>= 3.0.6, PHP 4)

Set the value of the /STATUS key

```
boolean 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 (PHP 3>= 3.0.6, PHP 4)

Get the value of the /STATUS key

```
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 (PHP 3>= 3.0.6, PHP 4)

Set the value of the /F key

```
boolean 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 (PHP 3>= 3.0.6, PHP 4)

Get the value of the /F key

```
string fdf_get_file (int fdf_document)
```

The **fdf_set_file()** returns the value of the /F key.

See also **fdf_set_file()**.

fdf_set_flags (PHP 4 >= 4.0.2)

Sets a flag of a field

```
boolean fdf_set_flags (int fdf_document, string fieldname, int whichFlags, int newFlags)
```

The **fdf_set_flags()** sets certain flags of the given field *fieldname*.

See also **fdf_set_opt()**.

fdf_set_opt (PHP 4 >= 4.0.2)

Sets an option of a field

```
boolean fdf_set_opt (int fdf_document, string fieldname, int element, string str1, string str2)
```

The **fdf_set_opt()** sets options of the given field *fieldname*.

See also **fdf_set_flags()**.

fdf_set_submit_form_action (PHP 4 >= 4.0.2)

Sets an javascript action of a field

```
boolean fdf_set_submit_form_action (int fdf_document, string fieldname, int trigger, string script, int flags)
```

The **fdf_set_submit_form_action()** sets a submit form action for the given field *fieldname*.

See also **fdf_set_javascript_action()**.

fdf_set_javascript_action (PHP 4 >= 4.0.2)

Sets an javascript action of a field

```
boolean fdf_set_javascript_action (int fdf_document, string fieldname, int trigger, string script)
```

The **fdf_set_javascript_action()** sets a javascript action for the given field *fieldname*.

See also **fdf_set_submit_form_action()**.

XXII. FTP functions

FTP stands for File Transfer Protocol.

The following constants are defined when using the FTP module: `FTP_ASCII` and `FTP_BINARY`.

Example 1. `ftp()` example

```
<?php
// set up basic connection
$conn_id = ftp_connect("$ftp_server");

// login with username and password
$login_result = ftp_login($conn_id, "$ftp_user_name", "$ftp_user_pass");

// check connection
if ((!$conn_id) || (!$login_result)) {
    echo "Ftp connection has failed!";
    echo "Attempted to connect to $ftp_server for user $user";
    die;
} else {
    echo "Connected to $ftp_server, for user $user";
}

// upload the file
$upload = ftp_put($conn_id, "$destination_file", "$source_file", FTP_BINARY);

// check upload status
if (!$upload) {
    echo "Ftp upload has failed!";
} else {
    echo "Uploaded $source_file to $ftp_server as $destination_file";
}

// close the FTP stream
ftp_quit($conn_id);
?>
```


ftp_connect (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Opens up an FTP connection

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Logs in an FTP connection

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the current directory name

```
string ftp_pwd (int ftp_stream)
```

Returns the current directory, or false on error.

ftp_cdup (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Changes to the parent directory

```
int ftp_cdup (int ftp_stream)
```

Returns true on success, false on error.

Changes to the parent directory.

ftp_chdir (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Changes directories on a FTP server

```
int ftp_chdir (int ftp_stream, string directory)
```

Returns true on success, false on error.

Changes to the specified *directory*.

ftp_mkdir (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Creates a directory

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Removes a directory

```
int ftp_rmdir (int ftp_stream, string directory)
```

Returns true on success, false on error.

Removes the specified *directory*.

ftp_nlist (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns a list of files in the given directory.

```
array ftp_nlist (int ftp_stream, string directory)
```

Returns an array of filenames on success, false on error.

ftp_rawlist (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns a detailed list of files in the given directory.

```
array 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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the system type identifier of the remote FTP server.

```
string ftp_systype (int ftp_stream)
```

Returns the remote system type, or false on error.

ftp_pasv (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Turns passive mode on or off.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Downloads a file from the FTP server.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Downloads a file from the FTP server and saves to an open file.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Uploads a file to the FTP server.

```
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**.

Example 1. Ftp_put() example

```
$upload = ftp_put ($conn_id, "$destination_file", "$source_file", FTP_ASCII);
```

ftp_fput (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Uploads from an open file to the FTP server.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the size of the given file.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the last modified time of the given file.

```
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.

Note: **ftp_mdtm()** does not work with directories.

ftp_rename (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Renames a file on the ftp server.

```
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 (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Deletes a file on the ftp server.

```
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_site (PHP 3>= 3.0.15, PHP 4 >= 4.0RC1)

Sends a SITE command to the server.

```
int ftp_site (int ftp_stream, string cmd)
```

Returns true on success, false on error.

ftp_site() sends the command specified by *cmd* to the FTP server. SITE commands are not standardized, and vary from server to server. They are useful for handling such things as file permissions and group membership.

ftp_quit (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Closes an FTP connection

```
int ftp_quit (int ftp_stream)
```

ftp_connect() closes *ftp_stream*.

XXIII. Function Handling functions

These functions all handle various operations involved in working with functions.

call_user_func (PHP 3>= 3.0.3, PHP 4)

Call a user function given by the first parameter

```
mixed call_user_func (string function_name [, mixed parameter [, mixed ...]])
```

Call a user defined function given by the *function_name* parameter. Take the following:

```
function barber ($type) {
    print "You wanted a $type haircut, no problem";
}
call_user_func ('barber', "mushroom");
call_user_func ('barber', "shave");
```

create_function (PHP 4 >= 4.0.1)

Create an anonymous (lambda-style) function

```
string create_function (string args, string code)
```

Creates an anonymous function from the parameters passed, and returns a unique name for it. Usually the *args* will be passed as a single quote delimited string, and this is also recommended for the *code*. The reason for using single quoted strings, is to protect the variable names from parsing, otherwise, if you use double quotes there will be a need to escape the variable names, e.g. \\${avar}.

You can use this function, to (for example) create a function from information gathered at run time:

Example 1. Creating an anonymous function with create_function()

```
$newfunc = create_function('$a,$b','return "ln($a) + ln($b) = ".log($a * $b);';
echo "New anonymous function: $newfunc\n";
echo $newfunc(2,M_E)."\n";
// outputs
// New anonymous function: lambda_1
// ln(2) + ln(2.718281828459) = 1.6931471805599
```

Or, perhaps to have general handler function that can apply a set of operations to a list of parameters:

Example 2. Making a general processing function with create_function()

```
function process($var1, $var2, $farr) {
    for ($f=0; $f < count($farr); $f++)
        echo $farr[$f]($var1,$var2)."\n";
}

// create a bunch of math functions
$f1 = 'if ($a >=0) {return "b*a^2 = ".$b*sqrt($a);} else {return false;}';
$f2 = "return \"min(b^2+a, a^2,b) = \".min(\$a*\$a+\$b,\$b*\$b+\$a);";
$f3 = 'if ($a > 0 && $b != 0) {return "ln(a)/b = ".log($a)/$b;} else {return false;}';
$farr = array(
    create_function('$x,$y', 'return "some trig: ".(sin($x) + $x*cos($y));'),
    create_function('$x,$y', 'return "a hypotenuse: ".sqrt($x*$x + $y*$y);'),
    create_function('$a,$b', $f1),
    create_function('$a,$b', $f2),
```

```

create_function('$a,$b', '$f3')
);

echo "\nUsing the first array of anonymous functions\n";
echo "parameters: 2.3445, M_PI\n";
process(2.3445, M_PI, $farr);

// now make a bunch of string processing functions
$garr = array(
    create_function('$b,$a','if (strcmp($a,$b,3) == 0) return "*** \"$a\" \"'.
        'and \"$b\"\\n** Look the same to me! (looking at the first 3 chars)\";'),
    create_function('$a,$b','; return "CRCs: ".crc32($a).", ".crc32(b);'),
    create_function('$a,$b','; return "similar(a,b) = ".similar_text($a,$b,&$p)."($p%)\";');
);
echo "\nUsing the second array of anonymous functions\n";
process("Twas brillig and the slithy toves", "Twas the night", $garr);

```

and when you run the code above, the output will be:

```

Using the first array of anonymous functions
parameters: 2.3445, M_PI
some trig: -1.6291725057799
a hypotenuse: 3.9199852871011
b*a^2 = 4.8103313314525
min(b^2+a, a^2,b) = 8.6382729035898
ln(a/b) = 0.27122299212594

Using the second array of anonymous functions
** "Twas the night" and "Twas brillig and the slithy toves"
** Look the same to me! (looking at the first 3 chars)
CRCs: -725381282 , 1908338681
similar(a,b) = 11(45.83333333333333%)

```

But perhaps the most common use for of lambda-style (anonymous) functions is to create callback functions, for example when using **array_walk()** or **usort()**

Example 3. Using anonymous functions as callback functions

```

$av = array("the ","a ","that ","this ");
array_walk($av, create_function('&$v,$k','$v = $v."mango";'));
print_r($av); // for PHP 3 use var_dump()
// outputs:
// Array
// (
//     [0] => the mango
//     [1] => a mango
//     [2] => that mango
//     [3] => this mango
// )

// an array of strings ordered from shorter to longer
$sv = array("small","larger","a big string","it is a string thing");
print_r($sv);
// outputs:
// Array
// (
//     [0] => small
//     [1] => larger
//     [2] => a big string
//     [3] => it is a string thing

```

```
// )

// sort it from longer to shorter
usort($sv, create_function('$a,$b','return strlen($b) - strlen($a);'));
print_r($sv);
// outputs:
// Array
// (
//     [0] => it is a string thing
//     [1] => a big string
//     [2] => larger
//     [3] => small
// )
```

func_get_arg (PHP 4 >= 4.0b4)

Return an item from the argument list

```
mixed func_get_arg (int arg_num)
```

Returns the argument which is at the *arg_num*'th offset into a user-defined function's argument list. Function arguments are counted starting from zero. **Func_get_arg()** will generate a warning if called from outside of a function definition.

If *arg_num* is greater than the number of arguments actually passed, a warning will be generated and **func_get_arg()** will return FALSE.

```
<?php
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br>\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg (1) . "<br>\n";
    }
}
foo (1, 2, 3);
?>
```

Func_get_arg() may be used in conjunction with **func_num_args()** and **func_get_args()** to allow user-defined functions to accept variable-length argument lists.

Note: This function was added in PHP 4.

func_get_args (PHP 4 >= 4.0b4)

Returns an array comprising a function's argument list

```
array func_get_args (void )
```

Returns an array in which each element is the corresponding member of the current user-defined function's argument list. **Func_get_args()** will generate a warning if called from outside of a function definition.

```
<?php
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br>\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg (1) . "<br>\n";
    }
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo "Argument $i is: " . $arg_list[$i] . "<br>\n";
    }
}
foo (1, 2, 3);
?>
```

Func_get_args() may be used in conjunction with **func_num_args()** and **func_get_arg()** to allow user-defined functions to accept variable-length argument lists.

Note: This function was added in PHP 4.

func_num_args (PHP 4 >= 4.0b4)

Returns the number of arguments passed to the function

```
int func_num_args (void )
```

Returns the number of arguments passed into the current user-defined function. **Func_num_args()** will generate a warning if called from outside of a function definition.

```
<?php
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs\n";
}
foo (1, 2, 3);      // Prints 'Number of arguments: 3'
?>
```

Func_num_args() may be used in conjunction with **func_get_arg()** and **func_get_args()** to allow user-defined functions to accept variable-length argument lists.

Note: This function was added in PHP 4.

function_exists (PHP 3>= 3.0.7, PHP 4)

Return true if the given function has been defined

```
bool function_exists (string function_name)
```

Checks the list of defined functions for *function_name*. Returns true if the given function name was found, false otherwise.

```
if (function_exists('imap_open')) {
    echo "IMAP functions are available.<br>\n";
} else {
    echo "IMAP functions are not available.<br>\n";
}
```

Note that a function name may exist, even if the function itself is unusable due to configuration or compiling options.

See also **method_exists()**.

register_shutdown_function (PHP 3>= 3.0.4, PHP 4)

Register a function for execution on shutdown

```
int register_shutdown_function (string func)
```

Registers the function named by *func* to be executed when script processing is complete.

Common Pitfalls:

Since no output is allowed to the browser in this function, you will be unable to debug it using statements such as print or echo.

XXIV. GNU Gettext

The gettext functions implement a NLS (Native Language Support) API which can be used to internationalize your PHP applications. Please see the GNU gettext documentation for a thorough explanation of these functions.

bindtextdomain (PHP 3>= 3.0.7, PHP 4)

Sets the path for a domain

```
string bindtextdomain (string domain, string directory)
```

The **bindtextdomain()** function sets the path for a domain.

dcgettext (PHP 3>= 3.0.7, PHP 4)

Overrides the domain for a single lookup

```
string dcgettext (string domain, string message, int category)
```

This function allows you to override the current domain for a single message lookup. It also allows you to specify a category.

dgettext (PHP 3>= 3.0.7, PHP 4)

Override the current domain

```
string dgettext (string domain, string message)
```

The **dgettext()** function allows you to override the current domain for a single message lookup.

gettext (PHP 3>= 3.0.7, PHP 4)

Lookup a message in the current domain

```
string gettext (string message)
```

This function returns a translated string if one is found in the translation table, or the submitted message if not found. You may use an underscore character as an alias to this function.

Example 1. Gettext()-check

```
<?php
// Set language to German
putenv ("LANG=de");

// Specify location of translation tables
bindtextdomain ("myPHPApp", "./locale");

// Choose domain
textdomain ("myPHPApp");

// Print a test message
print (gettext ("Welcome to My PHP Application"));
?>
```

textdomain (PHP 3>= 3.0.7, PHP 4)

Sets the default domain

```
string textdomain (string text_domain)
```

This function sets the domain to search within when calls are made to **gettext()**, usually the named after an application. The previous default domain is returned. Call it with no parameters to get the current setting without changing it.

XXV. GMP functions

These functions allow you to work with arbitrary-length integers using the GNU MP library. In order to have these functions available, you must compile PHP with GMP support by using the `-with-gmp` option.

You can download the GMP library from <http://www.swox.com/gmp/>. This site also has the GMP manual available.

You will need GMP version 2 or better to use these functions. Some functions may require more recent version of the GMP library.

These functions have been added in PHP 4.0.4.

Note: Most GMP functions accept GMP number arguments, defined as `resource` below. However, most of these functions will also accept numeric and string arguments, given that it is possible to convert the latter to a number. Also, if there is a faster function that can operate on integer arguments, it would be used instead of the slower function when the supplied arguments are integers. This is done transparently, so the bottom line is that you can use integers in every function that expects GMP number. See also the `gmp_init()` function.

Example 1. Factorial function using GMP

```
<?php
function fact ($x) {
    if ($x <= 1)
        return 1;
    else
        return gmp_mul ($x, fact ($x-1));
}
print gmp_strval (fact (1000)) . "\n";
?>
```

This will calculate factorial of 1000 (pretty big number) very fast.

gmp_init (unknown)

Create GMP number

```
resource gmp_init (mixed number)
```

Creates a GMP number from an integer or string. String representation can be decimal or hexadecimal. In the latter case, the string should start with 0x.

Example 1. Creating GMP number

```
<?php
    $a = gmp_init (123456);
    $b = gmp_init ("0xFFFFDEBACDFEDF7200");
?>
```

Note: It is not necessary to call this function if you want to use integer or string in place of GMP number in GMP functions, like **gmp_add()**. Function arguments are automatically converted to GMP numbers, if such conversion is possible and needed, using the same rules as **gmp_init()**.

gmp_intval (unknown)

Convert GMP number to integer

```
int gmp_intval (resource gmpnumber)
```

This function allows to convert GMP number to integer.

Warning

This function returns a useful result only if the number actually fits the PHP integer (i.e., signed long type). If you want just to print the GMP number, use **gmp_strval()**.

gmp_strval (unknown)

Convert GMP number to string

```
string gmp_strval (resource gmpnumber [, int base])
```

Convert GMP number to string representation in base *base*. The default base is 10. Allowed values for the base are from 2 to 36.

Example 1. Converting a GMP number to a string

```
<?php
    $a = gmp_init("0x41682179fbf5");
```

```
    printf ("Decimal: %s, 36-based: %s", gmp_strval($a), gmp_strval($a,36));
?>
```

gmp_add (unknown)

Add numbers

```
resource gmp_add (resource a, resource b)
```

Add two GMP numbers. The result will be a GMP number representing the sum of the arguments.

gmp_sub (unknown)

Subtract numbers

```
resource gmp_sub (resource a, resource b)
```

Subtracts *b* from *a* and returns the result.

gmp_mul (unknown)

Multiply numbers

```
resource gmp_mul (resource a, resource b)
```

Multiplies *a* by *b* and returns the result.

gmp_div_q (unknown)

Divide numbers

```
resource gmp_div_q (resource a, resource b [, int round])
```

Divides *a* by *b* and returns the integer result. The result rounding is defined by the *round*, which can have the following values:

- *GMP_ROUND_ZERO*: The result is truncated towards 0.
- *GMP_ROUND_PLUSINF*: The result is rounded towards +infinity.
- *GMP_ROUND_MINUSINF*: The result is rounded towards -infinity.

This function can also be called as **gmp_div()**.

See also **gmp_div_r()**, **gmp_div_qr()**

gmp_div_r (unknown)

Remainder of the division of numbers

```
resource gmp_div_r (resource n, resource d [, int round])
```

Calculates remainder of the integer division of *n* by *d*. The remainder has the sign of the *n* argument, if not zero.

See the **gmp_div_q()** function for description of the *round* argument.

See also **gmp_div_q()**, **gmp_div_qr()**

gmp_div_qr (unknown)

Divide numbers and get quotient and remainder

```
array gmp_div_qr (resource n, resource d [, int round])
```

The function divides *n* by *d* and returns array, with the first element being $[n/d]$ (the integer result of the division) and the second being $(n - [n/d] * d)$ (the remainder of the division).

See the **gmp_div_q()** function for description of the *round* argument.

Example 1. Division of GMP numbers

```
<?php
    $a = gmp_init ("0x41682179fbf5");
    $res = gmp_div_qr ($a, "0xDEFE75");
    printf("Result is: q - %s, r - %s",
           gmp_strval ($res[0]), gmp_strval ($res[1]));
?>
```

See also **gmp_div_q()**, **gmp_div_r()**.

gmp_div (unknown)

Divide numbers

```
resource gmp_divexact (resource a, resource b)
```

This function is an alias to **gmp_div_q()**.

gmp_mod (unknown)

Modulo operation

```
resource gmp_mod (resource n, resource d)
```

Calculates n modulo d . The result is always non-negative, the sign of d is ignored.

gmp_divexact (unknown)

Exact division of numbers

```
resource gmp_divexact (resource n, resource d)
```

Divides n by d , using fast "exact division" algorithm. This function produces correct results only when it is known in advance that d divides n .

gmp_cmp (unknown)

Compare numbers

```
int gmp_cmp (resource a, resource b)
```

Returns a positive value if $a > b$, zero if $a = b$ and negative value if $a < b$.

gmp_neg (unknown)

Negate number

```
resource gmp_neg (resource a)
```

Returns $-a$.

gmp_abs (unknown)

Absolute value

```
resource gmp_abs (resource a)
```

Returns absolute value of a .

gmp_sign (unknown)

Sign of number

```
int gmp_sign (resource a)
```

Return sign of a - 1 if a is positive and -1 if it's negative.

gmp_fact (unknown)

Factorial

```
resource gmp_fact (int a)
```

Calculates factorial ($a!$) of a .

gmp_sqrt (unknown)

Square root

```
resource gmp_sqrt (resource a)
```

Calculates square root of a .

gmp_sqrtrm (unknown)

Square root with remainder

```
array gmp_sqrtrm (resource a)
```

Returns array where first element is the integer square root of a (see also **gmp_sqrt()**), and the second is the remainder (i.e., the difference between a and the first element squared).

gmp_perfect_square (unknown)

Perfect square check

```
bool gmp_perfect_square (resource a)
```

Returns true if a is a perfect square, false otherwise.

See also: **gmp_sqrt()**, **gmp_sqrtrm()**.

gmp_pow (unknown)

Raise number into power

```
resource gmp_pow (resource base, int exp)
```

Raise $base$ into power exp . The case of 0^0 yields 1. exp cannot be negative.

gmp_powm (unknown)

Raise number into power with modulo

```
resource gmp_powm (resource base, resource exp, resource mod)
```

Calculate (*base* raised into power *exp*) modulo *mod*. If *exp* is negative, result is undefined.

gmp_prob_prime (unknown)

Check if number is "probably prime"

```
int gmp_prob_prime (resource a [, int reps])
```

If this function returns 0, *a* is definitely not prime. If it returns 1, then *a* is "probably" prime. If it returns 2, then *a* is surely prime. Reasonable values of *reps* vary from 5 to 10 (default being 10); a higher value lowers the probability for a non-prime to pass as a "probable" prime.

The function uses Miller-Rabin's probabilistic test.

gmp_gcd (unknown)

Calculate GCD

```
resource gmp_gcd (resource a, resource b)
```

Calculate greatest common divisor of *a* and *b*. The result is always positive even if either of, or both, input operands are negative.

gmp_gcdext (unknown)

Calculate GCD and multipliers

```
array gmp_gcdext (resource a, resource b)
```

Calculates *g*, *s*, and *t*, such that $a*s + b*t = g = \text{gcd}(a,b)$, where gcd is the greatest common divisor. Returns an array with respective elements *g*, *s* and *t*.

gmp_invert (unknown)

Inverse by modulo

```
resource gmp_invert (resource a, resource b)
```

Computes the inverse of *a* modulo *b*. Returns false if an inverse does not exist.

gmp_legendre (unknown)

Legendre symbol

```
int gmp_legendre (resource a, resource p)
```

Compute the Legendre symbol (<http://www.utm.edu/research/primes/glossary/LegendreSymbol.html>) of *a* and *p*. *p* should be odd and must be positive.

gmp_jacobi (unknown)

Jacobi symbol

```
int gmp_jacobi (resource a, resource p)
```

Computes Jacobi symbol (<http://www.utm.edu/research/primes/glossary/JacobiSymbol.html>) of *a* and *p*. *p* should be odd and must be positive.

gmp_random (unknown)

Random number

```
resource gmp_random (int limiter)
```

Generate a random number. The number will be up to *limiter* words long. If *limiter* is negative, negative numbers are generated.

gmp_and (unknown)

Logical AND

```
resource gmp_and (resource a, resource b)
```

Calculates logical AND of two GMP numbers.

gmp_or (unknown)

Logical OR

```
resource gmp_or (resource a, resource b)
```

Calculates logical inclusive OR of two GMP numbers.

gmp_xor (unknown)

Logical XOR

```
resource gmp_xor (resource a, resource b)
```

Calculates logical exclusive OR (XOR) of two GMP numbers.

gmp_setbit (unknown)

Set bit

```
resource gmp_setbit (resource &a, int index [, bool set_clear])
```

Sets bit *index* in *a*. *set_clear* defines if the bit is set to 0 or 1. By default the bit is set to 1.

gmp_clrbit (unknown)

Clear bit

```
resource gmp_clrbit (resource &a, int index)
```

Clears (sets to 0) bit *index* in *a*.

gmp_scan0 (unknown)

Scan for 0

```
int gmp_scan0 (resource a, int start)
```

Scans *a*, starting with bit *start*, towards more significant bits, until the first clear bit is found. Returns the index of the found bit.

gmp_scan1 (unknown)

Scan for 1

```
int gmp_scan1 (resource a, int start)
```

Scans *a*, starting with bit *start*, towards more significant bits, until the first set bit is found. Returns the index of the found bit.

gmp_popcount (unknown)

Population count

```
int gmp_popcount (resource a)
```

Return the population count of *a*.

gmp_hamdist (unknown)

Hamming distance

```
int gmp_hamdist (resource a, resource b)
```

Returns the hamming distance between *a* and *b*. Both operands should be non-negative.

XXVI. HTTP functions

These functions let you manipulate the output sent back to the remote browser right down to the HTTP protocol level.

header (PHP 3, PHP 4)

Send a raw HTTP header

```
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/rfc2616/rfc2616>) 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.

There are two special-case header calls. The first is the "Location" header. Not only does it send this header back to the browser, it also returns a REDIRECT status code to Apache. From a script writer's point of view this should not be important, but for people who understand Apache internals it is important to understand.

```
header ("Location: http://www.php.net"); /* Redirect browser
                                             to PHP web site */
exit;                                     /* Make sure that code below does
                                             not get executed when we redirect. */
```

The second special-case is any header that starts with the string, "HTTP/" (case is not significant). For example, if you have your ErrorDocument 404 Apache directive pointed to a PHP script, it would be a good idea to make sure that your PHP script is actually generating a 404. The first thing you do in your script should then be:

```
header ("HTTP/1.0 404 Not Found");
```

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
```

See also **headers_sent()**

headers_sent (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Returns true if headers have been sent

```
boolean headers_sent (void)
```

This function returns true if the HTTP headers have already been sent, false otherwise.

See also **header()**

setcookie (PHP 3, PHP 4)

Send a cookie

```
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.
- Cookies must be deleted with the same parameters as they were set with.

In PHP 3, multiple calls to **setcookie()** in the same script will be performed in reverse order. If you are trying to delete one cookie before inserting another you should put the insert before the delete. In PHP 4, multiple calls to **setcookie()** are performed in the order called.

Some examples follow how to send cookies:

Example 1. Setcookie() send examples

```
setcookie ("TestCookie", "Test Value");
setcookie ("TestCookie", $value,time()+3600); /* expire in 1 hour */
setcookie ("TestCookie", $value,time()+3600, "/~rasmus/", ".utoronto.ca", 1);
```

Examples follow how to delete cookies send in previous example:

Example 2. setcookie() delete examples

```
setcookie ("TestCookie");
// set the expiration date to one hour ago
setcookie ("TestCookie", "", time() - 3600);
setcookie ("TestCookie", "", time() - 3600, "/~rasmus/", ".utoronto.ca", 1);
```

When deleting a cookie you should assure that the expiration date is in the past, to trigger the removal mechanism in your browser.

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"];
```

You may also set array cookies by using array notation in the cookie name. This has the effect of setting as many cookies as you have array elements, but when the cookie is received by your script, the values are all placed in an array with the cookie's name:

```
setcookie ("cookie[three]", "cookiethree");
setcookie ("cookie[two]", "cookietwo");
setcookie ("cookie[one]", "cookieone");
if (isset ($cookie)) {
    while (list ($name, $value) = each ($cookie)) {
        echo "$name == $value<br>\n";
    }
}
```

For more information on cookies, see Netscape's cookie specification at
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.

XXVII. Hyperwave functions

Introduction

Hyperwave has been developed at IICM (<http://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/>). 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 is 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 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.com/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: A 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 (PHP 3>= 3.0.4, PHP 4)

convert attributes from object array to object record

```
string 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 (PHP 3>= 3.0.3, PHP 4)

object ids of children

```
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 (PHP 3>= 3.0.3, PHP 4)

object records of children

```
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 (PHP 3>= 3.0.3, PHP 4)

closes the Hyperwave connection

```
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 (PHP 3>= 3.0.3, PHP 4)

opens a connection

```
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 (PHP 3>= 3.0.3, PHP 4)

copies objects

```
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 (PHP 3>= 3.0.3, PHP 4)

deletes object

```
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 (PHP 3>= 3.0.3, PHP 4)

object id object belonging to anchor

```
int hw_docbyanchor (int connection, int anchorID)
```

Returns an th object id of the document to which *anchorID* belongs.

hw_DocByAnchorObj (PHP 3>= 3.0.3, PHP 4)

object record object belonging to anchor

```
string hw_docbyanchorobj (int connection, int anchorID)
```

Returns an th object record of the document to which *anchorID* belongs.

hw_Document_Attributes (PHP 3>= 3.0.3, PHP 4)

object record of hw_document

```
string hw_document_attributes (int hw_document)
```

Returns the object record of the document.

For backward compatibility, **hw_DocumentAttributes()** is also accepted. This is deprecated, however.

See also **hw_Document_BodyTag()**, and **hw_Document_Size()**.

hw_Document_BodyTag (PHP 3>= 3.0.3, PHP 4)

body tag of hw_document

```
string hw_document_bodytag (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_Document_Attributes()**, and **hw_Document_Size()**.

For backward compatibility, **hw_DocumentBodyTag()** is also accepted. This is deprecated, however.

hw_Document_Content (PHP 3>= 3.0.3, PHP 4)

returns content of hw_document

```
string hw_document_content (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_Document_Attributes()**, **hw_Document_Size()**, and **hw_DocumentSetContent()**.

hw_Document_SetContent (PHP 4 >= 4.0b2)

sets/replaces content of hw_document

```
string hw_document_setcontent (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 in the 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 functions fails the document will retain its old content.

See also **hw_Document_Attributes()**, **hw_Document_Size()**, and **hw_Document_Content()**.

hw_Document_Size (PHP 3>= 3.0.3, PHP 4)

size of hw_document

```
int hw_document_size (int hw_document)
```

Returns the size in bytes of the document.

See also **hw_Document_BodyTag()**, and **hw_Document_Attributes()**.

For backward compatibility, **hw_DocumentSize()** is also accepted. This is deprecated, however.

hw_ErrorMsg (PHP 3>= 3.0.3, PHP 4)

returns error message

```
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 (PHP 3>= 3.0.3, PHP 4)

retrieve text document

```
int hw_edittext (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_Document_BodyTag()**, **hw_Document_Size()**, **hw_Output_Document()**, **hw_GetText()**.

hw_Error (PHP 3>= 3.0.3, PHP 4)

error number

```
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 (PHP 3>= 3.0.3, PHP 4)

frees hw_document

```
int hw_free_document (int hw_document)
```

Frees the memory occupied by the Hyperwave document.

hw_GetParents (PHP 3>= 3.0.3, PHP 4)

object ids of parents

```
array hw_getparents (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 (PHP 3>= 3.0.3, PHP 4)

object records of parents

```
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 (PHP 3>= 3.0.3, PHP 4)

object ids of child collections

```
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()**, and **hw_GetChildDocColl()**.

hw_GetChildCollObj (PHP 3>= 3.0.3, PHP 4)

object records of child collections

```
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()**, and **hw_GetChildDocCollObj()**.

hw_GetRemote (PHP 3>= 3.0.3, PHP 4)

Gets a remote document

```
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 (PHP 3>= 3.0.3, PHP 4)

Gets children of remote document

```
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 (PHP 3>= 3.0.3, PHP 4)

Returns anchors pointing at object

```
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 (PHP 3>= 3.0.3, PHP 4)

object record

```
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()**, and **hw_GetObjectByQuery()**.

hw_GetAndLock (PHP 3>= 3.0.3, PHP 4)

return object record and lock object

```
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()**, and **hw_GetObject()**.

hw_GetText (PHP 3>= 3.0.3, PHP 4)

retrieve text document

```
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 . 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="[..../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_Document_BodyTag()**, **hw_Document_Size()**, and **hw_Output_Document()**.

hw_GetObjectByQuery (PHP 3>= 3.0.3, PHP 4)

search object

```
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 (PHP 3>= 3.0.3, PHP 4)

search object

```
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 (PHP 3>= 3.0.3, PHP 4)

search object in collection

```
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 (PHP 3>= 3.0.3, PHP 4)

search object in collection

```
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 (PHP 3>= 3.0.3, PHP 4)

object ids of child documents of collection

```
array hw_getchilddoccoll ( int connection, int objectID )
```

Returns array of object ids for child documents of a collection.

See also **hw_GetChildren()**, and **hw_GetChildColl()**.

hw_GetChildDocCollObj (PHP 3>= 3.0.3, PHP 4)

object records of child documents of collection

```
array hw_getchilddoccollobj ( int connection, int objectID )
```

Returns an array of object records for child documents of a collection.

See also **hw_ChildrenObj()**, and **hw_GetChildCollObj()**.

hw_GetAnchors (PHP 3>= 3.0.3, PHP 4)

object ids of anchors of document

```
array hw_getanchors ( int connection, int objectID )
```

Returns an array of object ids with anchors of the document with object ID *objectID*.

hw_GetAnchorsObj (PHP 3>= 3.0.3, PHP 4)

object records of anchors of document

```
array hw_getanchorsobj ( int connection, int objectID )
```

Returns an array of object records with anchors of the document with object ID *objectID*.

hw_Mv (PHP 3>= 3.0.3, PHP 4)

moves objects

```
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()**, and **hw_deleteobject()**.

hw_Identify (PHP 3>= 3.0.3, PHP 4)

identifies as user

```
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 (PHP 3>= 3.0.3, PHP 4)

check if object ids in collections

```
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 (PHP 3>= 3.0.3, PHP 4)

info about connection

```
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 (PHP 3>= 3.0.3, PHP 4)

insert collection

```
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 (PHP 3>= 3.0.3, PHP 4)

insert document

```
int hw_inscoll (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()**, and **hw_InsColl()**.

hw_InsertDocument (PHP 3>= 3.0.3, PHP 4)

upload any document

```
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 (PHP 3>= 3.0.3, PHP 4)

inserts an object record

```
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()**, and **hw_InsColl()**.

hw_mapid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Maps global id on virtual local id

```
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 (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

modifies object record

```
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 than 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 (PHP 3>= 3.0.3, PHP 4)

create new document

```
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 be passed in *document_size*. This function does not insert the document into the Hyperwave server.

See also **hw_FreeDocument()**, **hw_Document_Size()**, **hw_Document_BodyTag()**, **hw_Output_Document()**, and **hw_InsertDocument()**.

hw_Objrec2Array (PHP 3>= 3.0.3, PHP 4)

convert attributes from object record to object array

```
array hw_objrec2array (string object_record [, array format])
```

Converts an *object_record* into an object array. The keys of the resulting array are the attributes names. Multi-value 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. This left part must be two characters long. Other multi-value attributes without a prefix form an indexed array. If the optional parameter is missing the attributes 'Title', 'Description' and 'Keyword' are treated as language attributes and the attributes 'Group', 'Parent' and 'HtmlAttr' as non-prefixed multi-value attributes. By passing an array holding the type for each attribute you can alter this behaviour. The array is an associated array with the attribute name as its key and the value being one of HW_ATTR_LANG or HW_ATTR_NONE

See also **hw_array2objrec()**.

hw_Output_Document (PHP 3>= 3.0.3, PHP 4)

prints hw_document

```
int hw_output_document (int hw_document)
```

Prints the document without the BODY tag.

For backward compatibility, **hw_OutputDocument()** is also accepted. This is deprecated, however.

hw_pConnect (PHP 3>= 3.0.3, PHP 4)

make a persistent database connection

```
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 (PHP 3>= 3.0.3, PHP 4)

retrieve any document

```
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_Document_Size()**, **hw_Document_BodyTag()**, and **hw_Output_Document()**.

hw_Root (PHP 3>= 3.0.3, PHP 4)

root object id

```
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 (PHP 3>= 3.0.3, PHP 4)

unlock object

```
int hw_unlock (int connection, int objectID)
```

Unlocks a document, so other users regain access.

See also **hw_GetAndLock()**.

hw_Who (PHP 3>= 3.0.3, PHP 4)

List of currently logged in users

```
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_getusername (PHP 3>= 3.0.3, PHP 4)

name of currently logged in user

```
string hw_getusername (int connection)
```

Returns the username of the connection.

XXVIII. ICAP Functions

To get these functions to work, you have to compile PHP with `-with-icap`. That requires the icap library to be installed. Grab the latest version from <http://icap.chek.com/> and compile and install it.

icap_open (PHP 4 >= 4.0b4)

Opens up an ICAP connection

```
stream icap_open (string calendar, string username, string password, string options)
```

Returns an ICAP stream on success, false on error.

icap_open() opens up an ICAP connection to the specified *calendar* store. If the optional *options* is specified, passes the *options* to that mailbox also.

icap_close (unknown)

Close an ICAP stream

```
int icap_close (int icap_stream [, int flags])
```

Closes the given icap stream.

icap_fetch_event (PHP 4 >= 4.0b4)

Fetches an event from the calendar stream/

```
int icap_fetch_event (int stream_id, int event_id [, int options])
```

icap_fetch_event() fetches an event from the calendar stream specified by *event_id*.

Returns an event object consisting of:

- int id - ID of that event.
- int public - TRUE if the event is public, FALSE if it is private.
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - number of minutes before the event to send an alarm/reminder.
- object start - Object containing a datetime entry.
- object end - Object containing a datetime entry.

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour
- int min - minutes
- int sec - seconds

icap_list_events (PHP 4 >= 4.0RC1)

Return a list of events between two given datetimes

```
array icap_list_events ( int stream_id, int begin_date [, int end_date] )
```

Returns an array of event ID's that are between the two given datetimes.

Icap_list_events() function takes in a beginning datetime and an end datetime for a calendar stream. An array of event id's that are between the given datetimes are returned.

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour
- int min - minutes
- int sec - seconds

icap_store_event (PHP 4 >= 4.0b4)

Store an event into an ICAP calendar

```
string icap_store_event ( int stream_id, object event )
```

Icap_store_event() Stores an event into an ICAP calendar. An event object consists of:

- int public - 1 if public, 0 if private;
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - Number of minutes before the event to send out an alarm.
- datetime start - datetime object of the start of the event.
- datetime end - datetime object of the end of the event.

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour

- int min - minutes
- int sec - seconds

Returns true on success and false on error.

icap_delete_event (PHP 4 >= 4.0b4)

Delete an event from an ICAP calendar

```
string icap_delete_event (int stream_id, int uid)
```

Icap_delete_event() deletes the calendar event specified by the *uid*.

Returns true.

icap_snooze (PHP 4 >= 4.0b4)

Snooze an alarm

```
string icap_snooze (int stream_id, int uid)
```

Icap_snooze() turns on an alarm for a calendar event specified by the *uid*.

Returns true.

icap_list_alarms (PHP 4 >= 4.0b4)

Return a list of events that has an alarm triggered at the given datetime

```
int icap_list_alarms (int stream_id, array date, array time)
```

Returns an array of event ID's that has an alarm going off at the given datetime.

Icap_list_alarms() function takes in a datetime for a calendar stream. An array of event id's that has an alarm should be going off at the datetime are returned.

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour
- int min - minutes
- int sec - seconds

XXIX. Image functions

You can use the image functions in PHP to get the size of JPEG, GIF, PNG, and SWF 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.

The format of images you are able to manipulate depend on the version of gd you install, and any other libraries gd might need to access those image formats. Versions of gd older than gd-1.6 support gif format images, and do not support png, where versions greater than gd-1.6 support png, not gif.

In order to read and write images in jpeg format, you will need to obtain and install jpeg-6b (available at <ftp://ftp.uu.net/graphics/jpeg/>), and then recompile gd to make use of jpeg-6b. You will also have to compile PHP with `-with-jpeg-dir=/path/to/jpeg-6b`.

To add support for Type 1 fonts, you can install t1lib (available at <ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/software/t1lib/>), and then add `-with-t1lib[=dir]`.

GetImageSize (PHP 3, PHP 4)

Get the size of a GIF, JPEG, PNG or SWF image

```
array getimagesize (string filename [, array imageinfo])
```

The **GetImageSize()** function will determine the size of any GIF, JPG, PNG or SWF 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, 4 = SWF. 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 (file)

```
<?php $size = GetImageSize ("img/flag.jpg"); ?>
<IMG SRC="img/flag.jpg" <?php echo $size[3]; ?>
```

Example 2. GetImageSize (URL)

```
<?php $size = GetImageSize ("http://www.php.net/gifs/logo.gif"); ?>
```

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.iptc.org/> information in the APP13 marker. You can use the **iptcparse()** function to parse the binary APP13 marker into something readable.

Example 3. 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.

Note: URL support was added in PHP 4.0.5

ImageArc (PHP 3, PHP 4)

Draw a partial ellipse

```
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 (PHP 3, PHP 4)

Draw a character horizontally

```
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 **imageloadfont()**.

ImageCharUp (PHP 3, PHP 4)

Draw a character vertically

```
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 **imageloadfont()**.

ImageColorAllocate (PHP 3, PHP 4)

Allocate a color for an image

```
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);
```

ImageColorDeAllocate (PHP 3>= 3.0.6, PHP 4)

De-allocate a color for an image

```
int imagecolordeallocate (int im, int index)
```

The **ImageColorDeAllocate()** function de-allocates a color previously allocated with the **ImageColorAllocate()** function.

```
$white = ImageColorAllocate($im, 255, 255, 255);
ImageColorDeAllocate($im, $white);
```

ImageColorAt (PHP 3, PHP 4)

Get the index of the color of a pixel

```
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 (PHP 3, PHP 4)

Get the index of the closest color to the specified color

```
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 (PHP 3, PHP 4)

Get the index of the specified color

```
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 (PHP 3>= 3.0.2, PHP 4)

Get the index of the specified color or its closest possible alternative

```
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\(\)](#)**.

ImageGammaCorrect (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Apply a gamma correction to a GD image

```
int imagegammacorrect (int im, double inputgamma, double outputgamma)
```

The **ImageGammaCorrect()** function applies gamma correction to a gd image stream (*im*) given an input gamma, the parameter *inputgamma* and an output gamma, the parameter *outputgamma*.

ImageColorSet (PHP 3, PHP 4)

Set the color for the specified palette index

```
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 palettized images without the overhead of performing the actual flood-fill.

See also **[imagecolorat\(\)](#)**.

ImageColorsForIndex (PHP 3, PHP 4)

Get the colors for an index

```
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 (PHP 3, PHP 4)

Find out the number of colors in an image's palette

```
int imagecolorstotal (int im)
```

This returns the number of colors in the specified image's palette.

See also **[imagecolorat\(\)](#)** and **[imagecolorsforindex\(\)](#)**.

ImageColorTransparent (PHP 3, PHP 4)

Define a color as transparent

```
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.

ImageCopy (PHP 3>= 3.0.6, PHP 4)

Copy part of an image

```
int ImageCopy (int dst_im, int src_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h)
```

Copy a part of *src_im* onto *dst_im* starting at the x,y coordinates *src_x*, *src_y* with a width of *src_w* and a height of *src_h*. The portion defined will be copied onto the x,y coordinates, *dst_x* and *dst_y*.

ImageCopyResized (PHP 3, PHP 4)

Copy and resize part of an image

```
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 (PHP 3, PHP 4)

Create a new image

```
int imagecreate (int x_size, int y_size)
```

ImageCreate() returns an image identifier representing a blank image of size *x_size* by *y_size*.

Example 1. Creating a new GD image stream and outputting an image.

```
<?php
header ("Content-type: image/png");
$im = @ImageCreate (50, 100)
    or die ("Cannot Initialize new GD image stream");
$background_color = ImageColorAllocate ($im, 255, 255, 255);
$text_color = ImageColorAllocate ($im, 233, 14, 91);
```

```
ImageString ($im, 1, 5, 5, "A Simple Text String", $text_color);
ImagePng ($im);
?>
```

ImageCreateFromGIF (PHP 3, PHP 4)

Create a new image from file or URL

```
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@zymsys.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, $bgc);
        /* Output an errmsg */
        ImageString($im, 1, 5, 5, "Error loading $imgname", $tc);
    }
    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.

ImageCreateFromJPEG (PHP 3>= 3.0.16, PHP 4 >= 4.0RC1)

Create a new image from file or URL

```
int imagecreatefromjpeg (string filename)
```

ImageCreateFromJPEG() returns an image identifier representing the image obtained from the given filename.

ImageCreateFromJPEG() 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 JPEG:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```
function LoadJpeg ($imgname) {
    $im = @ImageCreateFromJPEG ($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, $bgc);
/* Output an errmsg */
ImageString ($im, 1, 5, 5, "Error loading $imgname", $tc);
}
return $im;
}

```

ImageCreateFromPNG (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Create a new image from file or URL

```
int imagecreatefrompng (string filename)
```

ImageCreateFromPNG() returns an image identifier representing the image obtained from the given filename.

ImageCreateFromPNG() 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 PNG:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```

function LoadPNG ($imgname) {
    $im = @ImageCreateFromPNG ($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, $bgc);
        /* Output an errmsg */
        ImageString ($im, 1, 5, 5, "Error loading $imgname", $tc);
    }
    return $im;
}

```

ImageCreateFromWBMP (PHP 4 >= 4.0.1)

Create a new image from file or URL

```
int imagecreatefromwbmp (string filename)
```

ImageCreateFromWBMP() returns an image identifier representing the image obtained from the given filename.

ImageCreateFromWBMP() 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 WBMP:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```
function LoadWBMP ($imgname) {
```

```

$im = @ImageCreateFromWBMP ($imgname); /* Attempt to open */
if (!$im) { /* See if it failed */
    $im = ImageCreate (20, 20); /* Create a blank image */
    $bgc = ImageColorAllocate ($im, 255, 255, 255);
    $tc = ImageColorAllocate ($im, 0, 0, 0);
    ImageFilledRectangle ($im, 0, 0, 10, 10, $bgc);
    /* Output an errmsg */
    ImageString ($im, 1, 5, 5, "Error loading $imgname", $tc);
}
return $im;
}

```

ImageCreateFromString (unknown)

Create a new image from the image stream in the string

```
int imagecreatefromstring (string image)
```

ImageCreateFromString() returns an image identifier representing the image obtained from the given string.

ImageDashedLine (PHP 3, PHP 4)

Draw a dashed line

```
int imagedashedline (int im, int x1, int y1, int x2, int y2, int col)
```

ImageDashedLine() 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 (PHP 3, PHP 4)

Destroy an image

```
int imagedestroy (int im)
```

ImageDestroy() frees any memory associated with image *im*. *Im* is the image identifier returned by the **ImageCreate()** function.

ImageFill (PHP 3, PHP 4)

Flood fill

```
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 (PHP 3, PHP 4)

Draw a filled polygon

```
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]* = *x*0, *points[1]* = *y*0, *points[2]* = *x*1, *points[3]* = *y*1, etc. *Num_points* is the total number of vertices.

ImageFilledRectangle (PHP 3, PHP 4)

Draw a filled rectangle

```
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 *x*1, *y*1 and ending at bottom right coordinates *x*2, *y*2. 0, 0 is the top left corner of the image.

ImageFillToBorder (PHP 3, PHP 4)

Flood fill to specific color

```
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 (PHP 3, PHP 4)

Get font height

```
int imagefontheight (int font)
```

Returns the pixel height of a character in the specified font.

See also **ImageFontWidth()** and **ImageLoadFont()**.

ImageFontWidth (PHP 3, PHP 4)

Get font width

```
int imagefontwidth (int font)
```

Returns the pixel width of a character in font.

See also **ImageFontHeight()** and **ImageLoadFont()**.

ImageGIF (PHP 3, PHP 4)

Output image to browser or file

```
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.

The following code snippet allows you to write more portable PHP applications by auto-detecting the type of GD support which is available. Replace the sequence `Header("Content-type: image/gif"); ImageGIF($im);` by the more flexible sequence:

```
<?php
    if (function_exists("imagegif")) {
        Header("Content-type: image/gif");
        ImageGIF($im);
    }
    elseif (function_exists("imagejpeg")) {
        Header("Content-type: image/jpeg");
        ImageJPEG($im, "", 0.5);
    }
    elseif (function_exists("imagepng")) {
        Header("Content-type: image/png");
        ImagePNG($im);
    }
    elseif (function_exists("imagewbmp")) {
        Header("Content-type: image/vnd.wap.wbmp");
        ImageWBMP($im);
    }
    else
        die("No image support in this PHP server");
?>
```

Note: As of version 3.0.18 and 4.0.2 you can use the function **imagetypes()** in place of **function_exists()** for checking the presence of the various supported image formats:

```
if (ImageTypes() & IMG_GIF) {
    Header("Content-type: image/gif");
    ImageGif($im);
}
elseif (ImageTypes() & IMG_JPG) {
    ... etc.
```

See also **ImagePNG()**, **ImageWBMP()**, **ImageJPEG()**, **ImageTypes()**.

ImagePNG (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Output a PNG image to either the browser or a file

```
int imagepng (int im [, string filename])
```

The **ImagePNG()** outputs a GD image stream (*im*) in PNG format to standard output (usually the browser) or, if a filename is given by the *filename* it outputs the image to the file.

```
<?php
$im = ImageCreateFromPng("test.png");
ImagePng($im);
?>
```

See also **ImageGIF()**, **ImageWBMP()**, **ImageJPEG()**, **ImageTypes()**.

ImageJPEG (PHP 3>= 3.0.16, PHP 4 >= 4.0RC1)

Output image to browser or file

```
int imagejpeg (int im [, string filename [, int quality]])
```

ImageJPEG() creates the JPEG file in filename from the image *im*. The *im* argument is the return from the **ImageCreate()** function.

The filename argument is optional, and if left off, the raw image stream will be output directly. To skip the filename argument in order to provide a quality argument just use an empty string (''). By sending an image/jpeg content-type using **header()**, you can create a PHP script that outputs JPEG images directly.

Note: JPEG support is only available in PHP if PHP was compiled against GD-1.8 or later.

See also **ImagePNG()**, **ImageGIF()**, **ImageWBMP()**, **ImageTypes()**.

ImageWBMP (PHP 3>= 3.0.15, PHP 4 >= 4.0.1)

Output image to browser or file

```
int imagewbmp (int im [, string filename])
```

ImageWBMP() creates the WBMP file in filename from the image *im*. The *im* argument is the return from the **ImageCreate()** function.

The filename argument is optional, and if left off, the raw image stream will be output directly. By sending an image/vnd.wap.wbmp content-type using **header()**, you can create a PHP script that outputs WBMP images directly.

Note: WBMP support is only available in PHP if PHP was compiled against GD-1.8 or later.

See also **ImagePNG()**, **ImageGIF()**, **ImageJPEG()**, **ImageTypes()**.

ImageInterlace (PHP 3, PHP 4)

Enable or disable interlace

```
int imageinterlace ( int im [, int interlace] )
```

ImageInterlace() turns the interlace bit on or off. If interlace is 1 the im image will be interlaced, and if interlace is 0 the interlace bit is turned off.

This functions returns whether the interlace bit is set for the image.

ImageLine (PHP 3, PHP 4)

Draw a line

```
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 (PHP 3, PHP 4)

Load a new font

```
int imageloadfont ( 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 (PHP 3, PHP 4)

Draw a polygon

```
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 (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Give the bounding box of a text rectangle using PostScript Type1 fonts

```
array imagepsbbox (string text, int font, int size [, int space [, int tightness [, 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 (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Change the character encoding vector of a font

```
int imagepsencodefont (int font_index, 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 (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Free memory used by a PostScript Type 1 font

```
void imagepsfreefont (int fontindex)
```

See also [ImagePSLoadFont\(\)](#).

ImagePSLoadFont (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Load a PostScript Type 1 font from file

```
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\(\)](#).

ImagePsExtendFont (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Extend or condense a font

```
bool imagepsextendfont (int font_index, double extend)
```

Extend or condense a font (*font_index*), if the value of the *extend* parameter is less than one you will be condensing the font.

ImagePsSlantFont (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

Slant a font

```
bool imagepsslantfont (int font_index, double slant)
```

Slant a font given by the *font_index* parameter with a slant of the value of the *slant* parameter.

ImagePSText (PHP 3>= 3.0.9, PHP 4 >= 4.0RC1)

To draw a text string over an image using PostScript Type1 fonts

```
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 (PHP 3, PHP 4)

Draw a rectangle

```
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 (PHP 3, PHP 4)

Set a single pixel

```
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 (PHP 3, PHP 4)

Draw a string horizontally

```
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 **ImageLoadFont()**.

ImageStringUp (PHP 3, PHP 4)

Draw a string vertically

```
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 **ImageLoadFont()**.

ImageSX (PHP 3, PHP 4)

Get image width

```
int imagesx (int im)
```

ImageSX() returns the width of the image identified by im .

See also **ImageCreate()** and **ImageSY()**.

ImageSY (PHP 3, PHP 4)

Get image height

```
int imagesy (int im)
```

ImageSY() returns the height of the image identified by im .

See also **ImageCreate()** and **ImageSX()**.

ImageTTFBox (PHP 3>= 3.0.1, PHP 4)

Give the bounding box of a text using TypeType fonts

```
array imagettfbbox (int size, int angle, string fontfile, string text)
```

This function calculates and returns the bounding box in pixels for a TrueType text.

text

The string to be measured.

size

The font size in pixels.

fontfile

The name of the TrueType font file. (Can also be an URL.)

angle

Angle in degrees in which *text* will be measured.

ImageTTFBox() 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 **ImageTTFFText()**.

ImageTTFFText (PHP 3, PHP 4)

Write text to the image using TrueType fonts

```
array imagettfftext (int im, int size, int angle, int x, int y, int col, string fontfile, string text)
```

ImageTTFFText() 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: {) 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, $white, "/path/arial.ttf",
    "Testing... Omega: &#937;");
ImageGif ($im);
ImageDestroy ($im);
?>
```

This function requires both the GD library and the FreeType (<http://www.freetype.org/>) library.

See also **ImageTTFBBox()**.

ImageTypes (PHP 4 >= 4.0.2)

Return the image types supported by this PHP build

```
int imagetypes(void);
```

This function returns a bit-field corresponding to the image formats supported by the version of GD linked into PHP. The following bits are returned, IMG_GIF | IMG_JPG | IMG_PNG | IMG_WBMP. To check for PNG support, for example, do this:

Example 1. ImageTypes

```
<?php
if (ImageTypes() & IMG_PNG) {
    echo "PNG Support is enabled";
}
?>
```

read_exif_data (PHP 4 >= 4.0.1)

Read the EXIF headers from a JPEG

```
array read_exif_data (string filename)
```

The **read_exif_data()** function reads the EXIF headers from a JPEG image file. It returns an associative array where the indexes are the Exif header names and the values are the values associated with those Exif headers. Exif headers tend to be present in JPEG images generated by digital cameras, but unfortunately each digital camera maker has a different idea of how to actually tag their images, so you can't always rely on a specific Exif header being present.

Example 1. **read_exif_data**

```
<?php
$exif = read_exif_data ('p0001807.jpg');
while(list($k,$v)=each($exif)) {
    echo "$k: $v<br>\n";
}
?>

Output:
FileName: p0001807.jpg
FileDateTime: 929353056
FileSize: 378599
CameraMake: Eastman Kodak Company
CameraModel: KODAK DC265 ZOOM DIGITAL CAMERA (V01.00)
DateTime: 1999:06:14 01:37:36
Height: 1024
Width: 1536
IsColor: 1
FlashUsed: 0
FocalLength: 8.0mm
RawFocalLength: 8
ExposureTime: 0.004 s (1/250)
RawExposureTime: 0.0040000001899898
ApertureFNumber: f/ 9.5
RawApertureFNumber: 9.5100002288818
FocusDistance: 16.66m
RawFocusDistance: 16.659999847412
Orientation: 1
ExifVersion: 0200
```

Note: This function is only available in PHP 4 compiled using **–enable-exif**

This function does not require the GD image library.

XXX. IMAP, POP3 and NNTP 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/libc-client.a` 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.

Note that these functions are not limited to the IMAP protocol, despite their name. The underlying c-client library also supports NNTP, POP3 and local mailbox access methods.

This document can't go into detail on all the topics touched by the provided functions. Further information is provided by the documentation of the c-client library source (`docs/internal.txt`) and the following RFC documents:

- RFC821 (<http://www.faqs.org/rfcs/rfc821.html>): Simple Mail Transfer Protocol (SMTP).
- RFC822 (<http://www.faqs.org/rfcs/rfc822.html>): Standard for ARPA internet text messages.
- RFC2060 (<http://www.faqs.org/rfcs/rfc2060.html>): Internet Message Access Protocol (IMAP) Version 4rev1.
- RFC1939 (<http://www.faqs.org/rfcs/rfc1939.html>): Post Office Protocol Version 3 (POP3).
- RFC977 (<http://www.faqs.org/rfcs/rfc977.html>): Network News Transfer Protocol (NNTP).
- RFC2076 (<http://www.faqs.org/rfcs/rfc2076.html>): Common Internet Message Headers.
- RFC2045 (<http://www.faqs.org/rfcs/rfc2045.html>), RFC2046 (<http://www.faqs.org/rfcs/rfc2046.html>), RFC2047 (<http://www.faqs.org/rfcs/rfc2047.html>), RFC2048 (<http://www.faqs.org/rfcs/rfc2048.html>) & RFC2049 (<http://www.faqs.org/rfcs/rfc2049.html>): Multipurpose Internet Mail Extensions (MIME).

A detailed overview is also available in the books *Programming Internet Email* (<http://www.oreilly.com/catalog/progintemail/noframes.html>) by David Wood and *Managing IMAP* (<http://www.oreilly.com/catalog/mimap/noframes.html>) by Dianna Mullet & Kevin Mullet.

imap_append (PHP 3, PHP 4)

Append a string message to a specified mailbox

```
int imap_append (int imap_stream, string mbox, string message [, string flags])
```

Returns true on sucess, false on error.

imap_append() appends a string message to the specified mailbox *mbox*. 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.

Example 1. **imap_append()** example

```
$stream = imap_open("{your imap host}INBOX.Drafts", "username", "password");

$check = imap_check($stream);
print "Msg Count before append: ". $check->Nmsgs . "\n";

imap_append($stream, "{your imap host}INBOX.Drafts"
            , "From: me@my.host\r\n"
            . "To: you@your.host\r\n"
            . "Subject: test\r\n"
            . "\r\n"
            . "this is a test message, please ignore\r\n"
            );

$check = imap_check($stream);
print "Msg Count after append : ". $check->Nmsgs . "\n";

imap_close($stream);
```

imap_base64 (PHP 3, PHP 4)

Decode BASE64 encoded text

```
string imap_base64 (string text)
```

imap_base64() function decodes BASE-64 encoded text (see RFC2045 (<http://www.faqs.org/rfcs/rfc2045.html>), Section 6.8). The decoded message is returned as a string.

See also **imap_binary()**.

imap_body (PHP 3, PHP 4)

Read the message body

```
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_body() will only return a verbatim copy of the message body. To extract single parts of a multipart MIME-encoded message you have to use **imap_fetchstructure()** to analyze its structure and **imap_fetchbody()** to extract a copy of a single body component.

imap_check (PHP 3, PHP 4)

Check current mailbox

```
object 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 - last change of mailbox contents
- Driver - protocol used to access this mailbox: POP3, IMAP, NNTP
- Mailbox - the mailbox name
- Nmsgs - number of messages in the mailbox
- Recent - number of recent messages in the mailbox

imap_close (PHP 3, PHP 4)

Close an IMAP stream

```
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, removing all messages marked for deletion.

imap_createmailbox (PHP 3, PHP 4)

Create a new mailbox

```
int imap_createmailbox (int imap_stream, string mbox)
```

imap_createmailbox() creates a new mailbox specified by *mbox*. Names containing international characters should be encoded by **imap_utf7_encode()**

Returns true on success and false on error.

See also **imap_renamemailbox()**, **imap_deletemailbox()** and **imap_open()** for the format of *mbox* names.

Example 1. imap_createmailbox() example

```
$mbox = imap_open("{your imap host}", "username", "password", OP_HALFOPEN)
|| die("can't connect: ".imap_last_error());

$name1 = "phpnewbox";
$name2 = imap_utf7_encode("phpnewb ");

$newname = $name1;

echo "Newname will be '$name1'<br>\n";

# we will now create a new mailbox "phptestbox" in your inbox folder,
# check its status after creation and finally remove it to restore
# your inbox to its initial state
if(@imap_createmailbox($mbox,imap_utf7_encode("{your imap host}INBOX.$newname")))
{
    $status = @imap_status($mbox,"{your imap host}INBOX.$newname",SA_ALL);
    if($status) {
        print("your new mailbox '$name1' has the following status:<br>\n");
        print("Messages: ". $status->messages ."<br>\n";
        print("Recent: ". $status->recent ."<br>\n";
        print("Unseen: ". $status->unseen ."<br>\n";
        print("UIDnext: ". $status->uidnext ."<br>\n";
        print("UIDvalidity:". $status->uidvalidity)."<br>\n";

        if(imap_renamemailbox($mbox,"{your imap host}INBOX.$newname","{your imap host}INBOX.$name2"))
            echo "renamed new mailbox from '$name1' to '$name2'<br>\n";
            $newname=$name2;
        } else {
            print "imap_renamemailbox on new mailbox failed: ".imap_last_error()."<br>\n";
        }
    } else {
        print "imap_status on new mailbox failed: ".imap_last_error()."<br>\n";
    }
    if(@imap_deletemailbox($mbox,"{your imap host}INBOX.$newname"))
        print "new mailbox removed to restore initial state<br>\n";
    } else {
        print "imap_deletemailbox on new mailbox failed: ".implode("<br>\n",imap_errors())."<br>\n";
    }

} else {
    print "could not create new mailbox: ".implode("<br>\n",imap_errors())."<br>\n";
}

imap_close($mbox);
```

imap_delete (PHP 3, PHP 4)

Mark a message for deletion from current mailbox

```
int imap_delete (int imap_stream, int msg_number [, int flags])
```

Returns true.

imap_delete() function marks message pointed by *msg_number* for deletion. The optional *flags* parameter only has a single option, *FT_UID*, which tells the function to treat the *msg_number* argument as a *UID*. Messages marked for deletion will stay in the mailbox until either **imap_expunge()** is called or **imap_close()** is called with the optional parameter *CL_EXPUNGE*.

Example 1. Imap_delete() Beispiel

```
$mbox = imap_open ("{your imap.host}INBOX", "username", "password")
|| die ("can't connect: " . imap_last_error ());

$check = imap_mailboxmsginfo ($mbox);
print "Messages before delete: " . $check->Nmsgs . "<br>\n";
imap_delete ($mbox, 1);
$check = imap_mailboxmsginfo ($mbox);
print "Messages after delete: " . $check->Nmsgs . "<br>\n";
imap_expunge ($mbox);
$check = imap_mailboxmsginfo ($mbox);
print "Messages after expunge: " . $check->Nmsgs . "<br>\n";
imap_close ($mbox);
```

imap_deletemailbox (PHP 3, PHP 4)

Delete a mailbox

```
int imap_deletemailbox (int imap_stream, string mbox)
```

imap_deletemailbox() deletes the specified mailbox (see **imap_open()** for the format of *mbox* names).

Returns true on success and false on error.

See also **imap_createmailbox()**, **imap_renamemailbox()**, and **imap_open()** for the format of *mbox*.

imap_expunge (PHP 3, PHP 4)

Delete all messages marked for deletion

```
int imap_expunge (int imap_stream)
```

imap_expunge() deletes all the messages marked for deletion by **imap_delete()**, **imap_mail_move()**, or **imap_setflag_full()**.

Returns true.

imap_fetchbody (PHP 3, PHP 4)

Fetch a particular section of the body of the message

```
string imap_fetchbody (int imap_stream, int msg_number, string part_number [, 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()** is a bitmask with one or more of the following:

- FT_UID - The *msg_number* is a UID
- FT_PEEK - Do not set the \Seen flag if not already set
- FT_INTERNAL - The return string is in "internal" format, without any attempt to canonicalize CRLF.

imap_fetchstructure (PHP 3, PHP 4)

Read the structure of a particular message

```
object imap_fetchstructure (int imap_stream, int msg_number [, int flags])
```

This function fetches all the structured information for a given message. The optional *flags* parameter only has a single option, *FT_UID*, which tells the function to treat the *msg_number* argument as a *UID*. The returned object includes the envelope, internal date, size, flags and body structure along with a similar object for each mime attachment. The structure of the returned objects is as follows:

Table 1. Returned Objects for imap_fetchstructure()

type	Primary body type
encoding	Body transfer encoding
ifsubtype	True if there is a subtype string
subtype	MIME subtype
ifdescription	True if there is a description string
description	Content description string
ifid	True if there is an identification string
id	Identification string
lines	Number of lines
bytes	Number of bytes
ifdisposition	True if there is a disposition string
disposition	Disposition string
ifdparameters	True if the dparameters array exists
dparameters	Disposition parameter array
ifparameters	True if the parameters array exists
parameters	MIME parameters array
parts	Array of objects describing each message part

1. dparameters is an array of objects where each object has an "attribute" and a "value" property.
2. Parameter is an array of objects where each object has an "attributte" and a "value" property.
3. Parts is an array of objects identical in structure to the top-level object, with the limitation that it cannot contain further 'parts' objects.

Table 2. Primary body type

0	text
1	multipart
2	message
3	application
4	audio
5	image
6	video
7	other

Table 3. Transfer encodings

0	7BIT
1	8BIT
2	BINARY
3	BASE64
4	QUOTED-PRINTABLE
5	OTHER

imap_headerinfo (PHP 3, PHP 4)

Read the header of the message

```
object imap_headerinfo (int imap_stream, int msg_number [, int fromlength [, int subjectlength [, string 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_header (PHP 3, PHP 4)

Read the header of the message

```
object imap_header (int imap_stream, int msg_number [, int fromlength [, int subjectlength [, string defaulthost] ]])
```

This is an alias to **imap_headerinfo()** and is identical to this in any way.

imap_rfc822_parse_headers (PHP 4 >= 4.0RC1)

Parse mail headers from a string

```
object imap_rfc822_parse_headers (string headers [, string defaulthost])
```

This function returns an object of various header elements, similar to **imap_header()**, except without the flags and other elements that come from the IMAP server.

imap_headers (PHP 3, PHP 4)

Returns headers for all messages in a mailbox

```
array imap_headers (int imap_stream)
```

Returns an array of string formatted with header info. One element per mail message.

imap_listmailbox (PHP 3, PHP 4)

Read the list of mailboxes

```
array imap_listmailbox (int imap_stream, string ref, string pattern)
```

Returns an array containing the names of the mailboxes. See **imap_getmailboxes()** for a description of *ref* and *pattern*.

Example 1. imap_listmailbox() example

```
$mbox = imap_open("{your imap host}", "username", "password", OP_HALFOPEN)
    || die("can't connect: ".imap_last_error());

$list = imap_listmailbox($mbox, "{your imap host}", "*");
if(is_array($list)) {
    reset($list);
    while (list($key, $val) = each($list))
        print imap_utf7_decode($val)."<br>\n";
} else
    print "imap_listmailbox failed: ".imap_last_error()."\\n";

imap_close($mbox);
```

imap_getmailboxes (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

Read the list of mailboxes, returning detailed information on each one

```
array imap_getmailboxes (int imap_stream, string ref, string pattern)
```

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.

Mailbox names containing international Characters outside the printable ASCII range will be encoded and may be decoded by **imap_utf7_decode()**.

ref should normally be just the server specification as described in **imap_open()**, and *pattern* specifies where in the mailbox hierarchy to start searching. If you want all mailboxes, pass '*' for *pattern*.

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; '~/mail/%' on UW_IMAPD will return every mailbox in the ~/mail directory, but none in subfolders of that directory.

Example 1. imap_getmailboxes() example

```
$mbox = imap_open("{your imap host}", "username", "password", OP_HALFOPEN)
    || die("can't connect: ".imap_last_error());

$list = imap_getmailboxes($mbox, "{your imap host}", "*");
if(is_array($list)) {
    reset($list);
    while (list($key, $val) = each($list))
    {
        print "($key) ";
        print imap_utf7_decode($val->name).",";
        print "'".$val->delimiter."',";
        print $val->attributes."<br>\n";
    }
} else
    print "imap_getmailboxes failed: ".imap_last_error()."\\n";

imap_close($mbox);
```

See also **imap_getsubscribed()**.

imap_listsubscribed (PHP 3, PHP 4)

List all the subscribed mailboxes

```
array imap_listsubscribed (int imap_stream, string ref, string pattern)
```

Returns an array of all the mailboxes that you have subscribed. This is almost identical to **imap_listmailbox()**, but will only return mailboxes the user you logged in as has subscribed.

imap_getsubscribed (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

List all the subscribed mailboxes

```
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 (PHP 3, PHP 4)

Copy specified messages to a mailbox

```
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 (as described in RFC2060 (<http://www.faqs.org/rfcs/rfc2060.html>)).

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 (PHP 3, PHP 4)

Move specified messages to a mailbox

```
int imap_mail_move (int imap_stream, string msglist, string mbox [, int flags])
```

Moves mail messages specified by *msglist* to specified mailbox. *msglist* is a range not just message numbers (as described in RFC2060 (<http://www.faqs.org/rfcs/rfc2060.html>)).

Flags is a bitmask and may contain the single option

- CP_UID - the sequence numbers contain UIDS

Returns true on success and false on error.

imap_num_msg (PHP 3, PHP 4)

Gives the number of messages in the current mailbox

```
int imap_num_msg (int imap_stream)
```

Return the number of messages in the current mailbox.

imap_num_recent (PHP 3, PHP 4)

Gives the number of recent messages in current mailbox

```
int imap_num_recent (int imap_stream)
```

Returns the number of recent messages in the current mailbox.

imap_open (PHP 3, PHP 4)

Open an IMAP stream to a mailbox

```
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, but some functions and features are not available on IMAP servers.

A mailbox name consists of a server part and a mailbox path on this server. The special name INBOX stands for the current users personal mailbox. The server part, which is enclosed in '{' and '}', consists of the servers name or ip address, a protocol secification (beginning with '/') and an optional port specifier beginnung with ':'. The server part is mandatory in all mailbox parameters. Mailbox names that contain international characters besides those in the printable ASCII space have to be encoded with **imap_utf7_encode()**.

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 .newsrc for news (NNTP only)
- OP_HALFOPEN - For IMAP and NNTP names, open a connection but dont open a mailbox
- CL_EXPUNGE - Expunge mailbox automatically upon mailbox close

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.

Example 1. imap_open() example

```
$mbox = imap_open ("{your imap.host:143}", "username", "password");

echo "<p><h1>Mailboxes</h1>\n";
$folders = imap_listmailbox ($mbox, "{your imap.host:143}", "*");

if ($folders == false) {
    echo "Call failed<br>\n";
} else {
    while (list ($key, $val) = each ($folders)) {
        echo $val."<br>\n";
    }
}

echo "<p><h1>Headers in INBOX</h1>\n";
$headers = imap_headers ($mbox);

if ($headers == false) {
    echo "Call failed<br>\n";
} else {
    while (list ($key,$val) = each ($headers)) {
        echo $val."<br>\n";
    }
}

imap_close($mbox);
```

imap_ping (PHP 3, PHP 4)

Check if the IMAP stream is still active

```
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. (As PHP scripts do not tend to run that long, i can hardly imagine that this function will be usefull to anyone.)

imap_renamemailbox (PHP 3, PHP 4)

Rename an old mailbox to new mailbox

```
int imap_renamemailbox (int imap_stream, string old_mbox, string new_mbox)
```

This function renames on old mailbox to new mailbox (see **imap_open()** for the format of *mbox* names).

Returns true on success and false on error.

See also **imap_createmailbox()**, **imap_deletemailbox()**, and **imap_open()** for the format of *mbox*.

imap_reopen (PHP 3, PHP 4)

Reopen IMAP stream to new mailbox

```
int imap_reopen (int imap_stream, string mailbox [, string flags])
```

This function reopens the specified stream to a new mailbox on an IMAP or NNTP server.

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 .newsrc for news (NNTP only)
- OP_HALFOPEN - For IMAP and NNTP names, open a connection but dont open a mailbox.
- CL_EXPUNGE - Expunge mailbox automatically upon mailbox close (see also **imap_delete()** and **imap_expunge()**)

Returns true on success and false on error.

imap_subscribe (PHP 3, PHP 4)

Subscribe to a mailbox

```
int imap_subscribe (int imap_stream, string mbox)
```

Subscribe to a new mailbox.

Returns true on success and false on error.

imap_undelete (PHP 3, PHP 4)

Unmark the message which is marked deleted

```
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()** or **imap_mail_move()**.

Returns true on success and false on error.

imap_unsubscribe (PHP 3, PHP 4)

Unsubscribe from a mailbox

```
int imap_unsubscribe (int imap_stream, string mbox)
```

Unsubscribe from a specified mailbox.

Returns true on success and false on error.

imap_qprint (PHP 3, PHP 4)

Convert a quoted-printable string to an 8 bit string

```
string imap_qprint (string string)
```

Convert a quoted-printable string to an 8 bit string (according to RFC2045 (<http://www.faqs.org/rfcs/rfc2045.html>), section 6.7).

Returns an 8 bit (binary) string.

See also **imap_8bit()**.

imap_8bit (PHP 3, PHP 4)

Convert an 8bit string to a quoted-printable string

```
string imap_8bit (string string)
```

Convert an 8bit string to a quoted-printable string (according to RFC2045 (<http://www.faqs.org/rfcs/rfc2045.html>), section 6.7).

Returns a quoted-printable string.

See also **imap_qprint()**.

imap_binary (PHP 3>= 3.0.2, PHP 4)

Convert an 8bit string to a base64 string

```
string imap_binary (string string)
```

Convert an 8bit string to a base64 string (according to RFC2045 (<http://www.faqs.org/rfcs/rfc2045.html>), Section 6.8).

Returns a base64 string.

See also **imap_base64()**.

imap_scanmailbox (PHP 3, PHP 4)

Read the list of mailboxes, takes a string to search for in the text of the mailbox

```
array imap_scanmailbox (int imap_stream, string ref, string pattern, string content)
```

Returns an array containing the names of the mailboxes that have *string* in the text of the mailbox. This function is similar to **imap_listmailbox()**, but it will additionally check for the presence of the string *content* inside the mailbox data. See **imap_getmailboxes()** for a description of *ref* and *pattern*.

imap_mailboxmsginfo (PHP 3>= 3.0.2, PHP 4)

Get information about the current mailbox

```
object 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. It is similar to **imap_status()**, but will additionally sum up the size of all messages in the mailbox, which will take some additional time to execute. It returns the information in an object with following properties.

Table 1. Mailbox properties

Date	date of last change
Driver	driver
Mailbox	name of the mailbox
Nmsgs	number of messages
Recent	number of recent messages
Unread	number of unread messages
Deleted	number of deleted messages
Size	mailbox size

Example 1. imap_mailboxmsginfo() example

```
<?php
```

```
$mbox = imap_open("{your imap host}INBOX", "username", "password")
|| die("can't connect: ".imap_last_error());

$check = imap_mailboxmsginfo($mbox);

if($check) {
    print "Date: " . $check->Date . "<br>\n";
    print "Driver: " . $check->Driver . "<br>\n";
    print "Mailbox: " . $check->Mailbox . "<br>\n";
    print "Messages: " . $check->Nmsgs . "<br>\n";
    print "Recent: " . $check->Recent . "<br>\n";
    print "Unread: " . $check->Unread . "<br>\n";
    print "Deleted: " . $check->Deleted . "<br>\n";
    print "Size: " . $check->Size . "<br>\n";
} else {
    print "imap_check() failed: ".imap_last_error(). "<br>\n";
}
imap_close($mbox);

?>
```

imap_rfc822_write_address (PHP 3>= 3.0.2, PHP 4)

Returns a properly formatted email address given the mailbox, host, and personal info.

```
string imap_rfc822_write_address (string mailbox, string host, string personal)
```

Returns a properly formatted email address as defined in RFC822 (<http://www.faqs.org/rfcs/rfc822.html>) given the mailbox, host, and personal info.

Example 1. imap_rfc822_write_address() example

```
print imap_rfc822_write_address("hartmut", "cvs.php.net", "Hartmut Holzgraefe")."\n";
```

imap_rfc822_parse_adrlist (PHP 3>= 3.0.2, PHP 4)

Parses an address string

```
array imap_rfc822_parse_adrlist (string address, string default_host)
```

This function parses the address string as defined in RFC822 (<http://www.faqs.org/rfcs/rfc822.html>) and for each address, returns an array of objects. The objects properties are:

- mailbox - the mailbox name (username)
- host - the host name

- personal - the personal name
- adl - at domain source route

Example 1. imap_rfc822_parse_adrlist() example

```
$address_string = "Hartmut Holzgraefe <hartmut@cvs.php.net>, postmas-
ter@somedomain.net, root";
$address_array = imap_rfc822_parse_adrlist($address_string, "somedomain.net");
if(! is_array($address_array)) die("somethings wrong\n");

reset($address_array);
while(list($key,$val)=each($address_array)){
    print "mailbox : ".$val->mailbox."<br>\n";
    print "host     : ".$val->host."<br>\n";
    print "personal: ".$val->personal."<br>\n";
    print "adl      : ".$val->adl."<p>\n";
}
```

imap_setflag_full (PHP 3>= 3.0.3, PHP 4)

Sets flags on messages

```
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 flags which you can set are "\\Seen", "\\Answered", "\\Flagged", "\\Deleted", "\\Draft", and "\\Recent" (as defined by RFC2060).

The options are a bit mask with one or more of the following:

ST_UID The sequence argument contains UIDs instead of
sequence numbers

Example 1. imap_setflag_full() example

```
$mbox = imap_open("{your imap host:143}", "username", "password")
|| die("can't connect: ".imap_last_error());

$status = imap_setflag_full($mbox, "2,5", "\\Seen \\Flagged");

print gettype($status)."\n";
print $status."\n";

imap_close($mbox);
```

imap_clearflag_full (PHP 3>= 3.0.3, PHP 4)

Clears flags on messages

```
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 flags which you can unset are "\Seen", "\Answered", "\Flagged", "\Deleted", "\Draft", and "\Recent" (as defined by RFC2060).

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 (PHP 3>= 3.0.3, PHP 4)

Sort an array of message headers

```
array imap_sort (int stream, int criteria, int reverse, int options)
```

Returns an array of message numbers sorted by the given parameters.

Reverse 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 (PHP 3>= 3.0.3, PHP 4)

Returns header for a message

```
string imap_fetchheader (int imap_stream, int msgno, int flags)
```

This function causes a fetch of the complete, unfiltered RFC822 (<http://www.faqs.org/rfcs/rfc822.html>) 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 (PHP 3>= 3.0.3, PHP 4)

This function returns the UID for the given message sequence number

```
int imap_uid (int imap_stream, int msgno)
```

This function returns the UID for the given message sequence number. An UID is an unique identifier that will not change over time while a message sequence number may change whenever the content of the mailbox changes. This function is the inverse of **imap_msgno()**.

imap_msgno (PHP 3>= 3.0.3, PHP 4)

This function returns the message sequence number for the given UID

```
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 (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

This function returns an array of messages matching the given search criteria

```
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 "joey 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 incomplete or inaccurate (see also RFC2060, section 6.4.4).

Valid values for flags are SE_UID, which causes the returned array to contain UIDs instead of messages sequence numbers.

imap_last_error (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

This function returns the last IMAP error (if any) that occurred during this page request

```
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 (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

This function returns all of the IMAP errors (if any) that have occurred during this page request or since the error stack was reset.

```
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 (PHP 3>= 3.0.12, PHP 4 >= 4.0b4)

This function returns all IMAP alert messages (if any) that have occurred during this page request or since the alert stack was reset

```
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 (PHP 3>= 3.0.4, PHP 4)

This function returns status information on a mailbox other than the current one

```
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.

Example 1. imap_status() example

```
$mbox = imap_open("{your imap host}", "username", "password", OP_HALFOPEN)
    || die("can't connect: ".imap_last_error());

$status = imap_status($mbox, "{your imap host}INBOX", SA_ALL);
if($status) {
    print("Messages: ". $status->messages . "<br>\n";
    print("Recent: ". $status->recent . "<br>\n";
    print("Unseen: ". $status->unseen . "<br>\n";
    print("UIDnext: ". $status->uidnext . "<br>\n";
    print("UIDvalidity:". $status->uidvalidity). "<br>\n";
} else
    print "imap_status failed: ".imap_lasterror(). "\n";

imap_close($mbox);
```

imap_utf7_decode (PHP 3>= 3.0.15, PHP 4 >= 4.0b4)

Decodes a modified UTF-7 encoded string.

```
string imap_utf7_decode (string text)
```

Decodes modified UTF-7 *text* into 8bit data.

Returns the decoded 8bit data, or false if the input string was not valid modified UTF-7. This function is needed to decode mailbox names that contain international characters outside of the printable ASCII range. The modified UTF-7 encoding is defined in RFC 2060 (<http://www.faqs.org/rfcs/rfc2060.html>), section 5.1.3 (original UTF-7 was defined in RFC1642 (<http://www.faqs.org/rfcs/rfc1642.html>)).

imap_utf7_encode (PHP 3>= 3.0.15, PHP 4 >= 4.0b4)

Converts 8bit data to modified UTF-7 text.

```
string imap_utf7_encode (string data)
```

Converts 8bit *data* to modified UTF-7 text. This is needed to encode mailbox names that contain international characters outside of the printable ASCII range. The modified UTF-7 encoding is defined in RFC 2060 (<http://www.faqs.org/rfcs/rfc2060.html>), section 5.1.3 (original UTF-7 was defined in RFC1642 (<http://www.faqs.org/rfcs/rfc1642.html>)).

Returns the modified UTF-7 text.

imap_utf8 (PHP 3>= 3.0.13, PHP 4 >= 4.0RC1)

Converts text to UTF8

```
string imap_utf8 (string text)
```

Converts the given *text* to UTF8 (as defined in RFC2044 (<http://www.faqs.org/rfcs/rfc2044.html>)).

imap_fetch_overview (PHP 3>= 3.0.4, PHP 4)

Read an overview of the information in the headers of the given message

```
array imap_fetch_overview (int imap_stream, string sequence [, int flags])
```

This function fetches mail headers for the given *sequence* and returns an overview of their contents. *sequence* will contain a sequence of message indices or UIDs, if *flags* contains FT_UID. The returned value is an array of objects describing one message header each:

- subject - the messages subject
- from - who sent it

- date - when was it sent
- message_id - Message-ID
- references - is a reference to this message id
- size - size in bytes
- uid - UID the message has in the mailbox
- msgno - message sequence number in the mailbox
- recent - this message is flagged as recent
- flagged - this message is flagged
- answered - this message is flagged as answered
- deleted - this message is flagged for deletion
- seen - this message is flagged as already read
- draft - this message is flagged as being a draft

Example 1. imap_fetch_overview() example

```
$mbox = imap_open( "{your imap host:143}" , "username" , "password" )
|| die("can't connect: ".imap_last_error());

$overview = imap_fetch_overview($mbox, "2,4:6",0);

if(is_array($overview)) {
    reset($overview);
    while( list($key,$val) = each($overview)) {
        print      $val->msgno
        . " - " . $val->date
        . " - " . $val->subject
        . "\n";
    }
}

imap_close($mbox);
```

imap_mime_header_decode (PHP 3>= 3.0.17, PHP 4 >= 4.0RC1)

Decode MIME header elements

```
array imap_header_decode (string text)
```

imap_mime_header_decode() function decodes MIME message header extensions that are non ASCII text (see RFC2047 (<http://www.faqs.org/rfcs/rfc2047.html>)) The decoded elements are returned in an array of objects, where each object has two properties, "charset" & "text". If the element hasn't been encoded, and in other words is in plain US-ASCII, the "charset" property of that element is set to "default".

Example 1. imap_mime_header_decode() example

```
$text="=?ISO-8859-1?Q?Keld_J=F8rn_Simonsen?= <keld@dkuug.dk> ;
```

```
$elements=imap_mime_header_decode($text);
for($i=0;$i<count($elements);$i++) {
echo "Charset: {$elements[$i]->charset}\n";
echo "Text: {$elements[$i]->text}\n\n";
}
```

In the above example we would have two elements, whereas the first element had previously been encoded with ISO-8859-1, and the second element would be plain US-ASCII.

imap_mail_compose (PHP 3>= 3.0.5, PHP 4)

Create a MIME message based on given envelope and body sections

```
string imap_mail_compose (array envelope, array body)
```

Example 1. imap_mail_compose() example

```
<?php

$envelope[ "from" ]="musone@afterfive.com";
$envelope[ "to" ]="musone@darkstar";
$envelope[ "cc" ]="musone@edgeglobal.com";

$part1[ "type" ]=TYPEMULTIPART;
$part1[ "subtype" ]="mixed";

$filename="/tmp/imap.c.gz";
$fp=fopen($filename,"r");
$contents=fread($fp,filesize($filename));
fclose($fp);

$part2[ "type" ]=TYPEAPPLICATION;
$part2[ "encoding" ]=ENCBINARIES;
$part2[ "subtype" ]="octet-stream";
$part2[ "description" ]=basename($filename);
$part2[ "contents.data" ]=$contents;

$part3[ "type" ]=TYPETEXT;
$part3[ "subtype" ]="plain";
$part3[ "description" ]="description3";
$part3[ "contents.data" ]="contents.data3\n\n\n\t";

$body[1]=$part1;
$body[2]=$part2;
$body[3]=$part3;

echo nl2br(imap_mail_compose($envelope,$body));

?>
```

imap_mail (PHP 3>= 3.0.14, PHP 4 >= 4.0b4)

Send an email message

```
string imap_mail (string to, string subject, string message [, string additional_headers [, string cc [, string bcc [, string xpath]]]])
```

This function is currently only available in PHP 3.

XXXI. Informix functions

The Informix driver for Informix (IDS) 7.x, SE 7.x, Universal Server (IUS) 9.x and IDS 2000 is implemented in "ifx.ec" and "php3_ifx.h" in the informix extension directory. IDS 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: You need a version of ESQL/C to compile the PHP Informix driver. ESQL/C versions from 7.2x on should be OK. ESQL/C is now part of the Informix Client SDK.

Make sure that the "INFORMIXDIR" variable has been set, and that \$INFORMIXDIR/bin is in your PATH before you run the "configure" script.

The configure script will autodetect the libraries and include directories, if you run "configure --with_informix=yes". You can overide 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.

Runtime considerations: Make sure that the Informix environment variables INFORMIXDIR and INFORMIXSERVER are available to the PHP ifx driver, and that the INFORMIX bin directory is in the PATH. Check this by running a script that contains a call to `phpinfo()` before you start testing. The `phpinfo()` output should list these environment variables. This is true for both CGI php and Apache mod_php. You may have to set these environment variables in your Apache startup script.

The Informix shared libraries should also be available to the loader (check LD_LIBRARY_PATH or ld.so.conf/ldconfig).

Some notes on the use of BLOBs (TEXT and BYTE columns): BLOBs are normally addressed by 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, so that you don't need to use blob id's for select queries.

ifx_byteasvarchar(1) : return BYTE columns as if they were VARCHAR columns, so that you don't need to use 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, to save you some "chopping".

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 (PHP 3>= 3.0.3, PHP 4)

Open Informix server connection

```
int ifx_connect ([string database [, string userid [, string password] ]])
```

Returns a 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_connect ("mydb@ol_srv1", "imyself", "mypassword");
```

ifx_pconnect (PHP 3>= 3.0.3, PHP 4)

Open persistent Informix connection

```
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 (PHP 3>= 3.0.3, PHP 4)

Close Informix connection

```
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 (PHP 3>= 3.0.3, PHP 4)

Send Informix query

```
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.

A "result_id" resource 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 bitmask and can be either IFX_SCROLL, IFX_HOLD, or both or'ed together. Non-select queries are "execute immediate". IFX_SCROLL and IFX_HOLD are symbolic constants and as such shouldn't be between quotes. If you omit this parameter the cursor is a normal sequential cursor.

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 (PHP 3>= 3.0.4, PHP 4)

Prepare an SQL-statement for execution

```
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 bitmask 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 (PHP 3>= 3.0.4, PHP 4)

Execute a previously prepared SQL-statement

```
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 (PHP 3>= 3.0.3, PHP 4)

Returns error code of last Informix call

```
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 (PHP 3>= 3.0.4, PHP 4)

Returns error message of last Informix call

```
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 (PHP 3>= 3.0.3, PHP 4)

Get number of rows affected by a query

```
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. The database server can never return the actual number of rows that will be returned by a SELECT because it has not even begun fetching them at this stage (just after the "PREPARE" when the optimizer has determined the query plan).

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 (PHP 3>= 3.0.8, PHP 4)

Get the contents of sqlca.sqlerrd[0..5] after a query

```
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] ... 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 column') ", $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 (PHP 3>= 3.0.3, PHP 4)

Get row as enumerated array

```
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 SCROLL cursors.

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, with the column name as key.

Subsequent calls 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 (PHP 3>= 3.0.3, PHP 4)

Formats all rows of a query into a HTML table

```
int ifx_htmtbl_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_htmtbl_result ($rid, "border=\"2\"");
ifx_free_result($rid);
```

ifx_fieldtypes (PHP 3>= 3.0.3, PHP 4)

List of Informix SQL fields

```
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. Fielnames 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 (PHP 3>= 3.0.3, PHP 4)

List of SQL fieldproperties

```
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

```
$properties = ifx_fieldproperties ($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 (PHP 3>= 3.0.3, PHP 4)

Returns the number of columns in the query

```
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 (PHP 3>= 3.0.3, PHP 4)

Count the rows already fetched a query

```
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 (PHP 3>= 3.0.3, PHP 4)

Releases resources for the query

```
int ifx_free_result (int result_id)
```

Releases resources for the query associated with *result_id*. Returns FALSE on error.

ifx_create_char (PHP 3>= 3.0.6, PHP 4)

Creates an char object

```
int ifx_create_char (string param)
```

Creates an char object. *param* should be the char content.

ifx_free_char (PHP 3>= 3.0.6, PHP 4)

Deletes the char object

```
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 (PHP 3>= 3.0.6, PHP 4)

Updates the content of the char object

```
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 (PHP 3>= 3.0.6, PHP 4)

Return the content of the char object

```
int ifx_get_char (int bid)
```

Returns the content of the char object for the given char object-id *bid*.

ifx_create_blob (PHP 3>= 3.0.4, PHP 4)

Creates an blob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Duplicates the given blob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Deletes the blob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Return the content of a blob object

```
int ifx_get_blob (int bid)
```

Returns the content of the blob object for the given blob object-id *bid*.

ifx_update_blob (PHP 3>= 3.0.4, PHP 4)

Updates the content of the blob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Set the default blob mode for all select queries

```
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 (PHP 3>= 3.0.4, PHP 4)

Set the default text mode

```
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 (PHP 3>= 3.0.4, PHP 4)

Set the default byte mode

```
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 (PHP 3>= 3.0.4, PHP 4)

Sets the default return value on a fetch row

```
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 (PHP 3>= 3.0.4, PHP 4)

Creates an slob object and opens it

```
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.

ifxus_free_slob (PHP 3>= 3.0.4, PHP 4)

Deletes the slob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Deletes the slob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Opens an slob object

```
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 (PHP 3>= 3.0.4, PHP 4)

Returns the current file or seek position

```
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 (PHP 3>= 3.0.4, PHP 4)

Sets the current file or seek position

```
int ifxus_seek_blob (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 an byte offset. Return FALSE on error otherwise the seek position.

ifxus_read_slob (PHP 3>= 3.0.4, PHP 4)

Reads nbytes of the slob object

```
int ifxus_read_slob (long bid, long nbytes)
```

Reads nbytes of the slob object. *bid* is a existing slob id and *nbytes* is the number of bytes zu read. Return FALSE on error otherwise the string.

ifxus_write_slob (PHP 3>= 3.0.4, PHP 4)

Writes a string into the blob object

```
int ifxus_write_slob (long bid, string content)
```

Writes a string into the blob object. *bid* is a existing blob id and *content* the content to write. Return FALSE on error otherwise bytes written.

XXXII. InterBase functions

InterBase is a popular database put out by Borland/Inprise. More information about InterBase is available at <http://www.interbase.com/>. Oh, by the way, InterBase just joined the open source movement!

Note: Full support for InterBase 6 was added in PHP 4.0.

This database uses a single quote (') character for escaping, a behavior similar to the Sybase database, add to your `php.ini` the following directive:

```
magic_quotes_sybase = On
```


ibase_connect (PHP 3>= 3.0.6, PHP 4)

Open a connection to an InterBase database

```
int ibase_connect (string database [, string username [, string password [, string charset [, int buffers [, int dialect [, string role]]]]])
```

Establishes a connection to an InterBase server. The *database* argument has to be a valid path to database file on the server it resides on. If the server is not local, it must be prefixed with either 'hostname:' (TCP/IP), '//hostname/' (NetBEUI) or 'hostname@' (IPX/SPX), depending on the connection protocol used. *username* and *password* can also be specified with PHP configuration directives *ibase.default_user* and *ibase.default_password*. *charset* is the default character set for a database. *buffers* is the number of database buffers to allocate for the server-side cache. If 0 or omitted, server chooses its own default. *dialect* selects the default SQL dialect for any statement executed within a connection, and it defaults to the highest one supported by client libraries.

In case a second call is made to **ibase_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 **ibase_close()**.

Example 1. Ibase_connect() example

```
<?php
$dbh = ibase_connect ($host, $username, $password);
$stmt = 'SELECT * FROM tblname';
$sth = ibase_query ($dbh, $stmt);
while ($row = ibase_fetch_object ($sth)) {
    print $row->email . "\n";
}
ibase_close ($dbh);
?>
```

Note: *buffers* was added in PHP4-RC2.

Note: *dialect* was added in PHP4-RC2. It is functional only with InterBase 6 and versions higher than that.

Note: *role* was added in PHP4-RC2. It is functional only with InterBase 5 and versions higher than that.

See also: **ibase_pconnect()**.

ibase_pconnect (PHP 3>= 3.0.6, PHP 4)

Creates an persistent connection to an InterBase database

```
int ibase_pconnect (string database [, string username [, string password [, string charset [, int buffers [, int dialect [, string role]]]]])
```

ibase_pconnect() acts very much like **ibase_connect()** with two major differences. First, when connecting, the function will first try to find a (persistent) link that's already opened with the same parameters. If one is found, an identifier for it will be returned instead of opening a new connection. Second, the connection to the InterBase server will not be closed

when the execution of the script ends. Instead, the link will remain open for future use (**ibase_close()**) will not close links established by **ibase_pconnect()**). This type of link is therefore called 'persistent'.

Note: *buffers* was added in PHP4-RC2.

Note: *dialect* was added in PHP4-RC2. It is functional only with InterBase 6 and versions higher than that.

Note: *role* was added in PHP4-RC2. It is functional only with InterBase 5 and versions higher than that.

See also **ibase_connect()** for the meaning of parameters passed to this function. They are exactly the same.

ibase_close (PHP 3>= 3.0.6, PHP 4)

Close a connection to an InterBase database

```
int ibase_close ([int connection_id])
```

Closes the link to an InterBase database that's associated with a connection id returned from **ibase_connect()**. If the connection id is omitted, the last opened link is assumed. Default transaction on link is committed, other transactions are rolled back.

ibase_query (PHP 3>= 3.0.6, PHP 4)

Execute a query on an InterBase database

```
int ibase_query ([int link_identifier, string query [, int bind_args]])
```

Performs a query on an InterBase database, returning a result identifier for use with **ibase_fetch_row()**, **ibase_fetch_object()**, **ibase_free_result()** and **ibase_free_query()**.

Note: Although this function supports variable binding to parameter placeholders, there is not very much meaning using this capability with it. For real life use and an example, see **ibase_prepare()** and **ibase_execute()**.

ibase_fetch_row (PHP 3>= 3.0.6, PHP 4)

Fetch a row from an InterBase database

```
array ibase_fetch_row (int result_identifier)
```

Returns the next row specified by the result identifier obtained using the **ibase_query()**.

ibase_fetch_object (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Get an object from a InterBase database

```
object ibase_fetch_object (int result_id)
```

Fetches a row as a pseudo-object from a *result_id* obtained either by **ibase_query()** or **ibase_execute()**.

```
<php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);
    while ($row = ibase_fetch_object ($sth)) {
        print $row->email . "\n";
    }
    ibase_close ($dbh);
?>
```

See also **ibase_fetch_row()**.

ibase_field_info (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Get information about a field

```
array ibase_field_info (int result, int field number)
```

Returns an array with information about a field after a select query has been run. The array is in the form of name, alias, relation, length, type.

```
// helio@helio.com.br 08-Dec-2000 02:53

$rs=ibase_query("Select * from something");
$coln = ibase_num_fields($rs);
for ($i=0 ; $i < $coln ; $i++) {
    $col_info = ibase_field_info($rs, $i);
    echo "name: ".$col_info['name']."\n";
    echo "alias: ".$col_info['alias']."\n";
    echo "relation: ".$col_info['relation']."\n";
    echo "length: ".$col_info['length']."\n";
    echo "type: ".$col_info['type']."\n";
}
```

ibase_free_result (PHP 3>= 3.0.6, PHP 4)

Free a result set

```
int ibase_free_result (int result_identifier)
```

Free's a result set the has been created by **ibase_query()**.

ibase_prepare (PHP 3>= 3.0.6, PHP 4)

Prepare a query for later binding of parameter placeholders and execution

```
int ibase_prepare ([int link_identifier, string query])
```

Prepare a query for later binding of parameter placeholders and execution (via **ibase_execute()**).

ibase_execute (PHP 3>= 3.0.6, PHP 4)

Execute a previously prepared query

```
int ibase_execute (int query [, int bind_args])
```

Execute a query prepared by **ibase_prepare()**. This is a lot more effective than using **ibase_query()** if you are repeating a same kind of query several times with only some parameters changing.

```
<?php
    $updates = array(
        1 => 'Eric',
        5 => 'Filip',
        7 => 'Larry'
    );
    $query = ibase_prepare ("UPDATE FOO SET BAR = ? WHERE BAZ = ?");

    while (list($baz, $bar) = each($updates)) {
        ibase_execute($query, $bar, $baz);
    }
?>
```

ibase_trans (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Begin a transaction

```
int ibase_trans ([int trans_args [, int link_identifier]])
```

Begins a transaction.

ibase_commit (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Commit a transaction

```
int ibase_commit ([int link_identifier, int trans_number])
```

Commits transaction *trans_number* which was created with **ibase_trans()**.

ibase_rollback (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Rolls back a transaction

```
int ibase_rollback ([int link_identifier, int trans_number])
```

Rolls back transaction *trans_number* which was created with **ibase_trans()**.

ibase_free_query (PHP 3>= 3.0.6, PHP 4)

Free memory allocated by a prepared query

```
int ibase_free_query (int query)
```

Free a query prepared by **ibase_prepare()**.

ibase_timefmt (PHP 3>= 3.0.6, PHP 4)

Sets the format of timestamp, date and time type columns returned from queries

```
int ibase_timefmt (string format [, int columntype])
```

Sets the format of timestamp, date or time type columns returned from queries. Internally, the columns are formatted by c-function strftime(), so refer to it's documentation regarding to the format of the string. *columntype* is one of the constants IBASE_TIMESTAMP, IBASE_DATE and IBASE_TIME. If omitted, defaults to IBASE_TIMESTAMP for backwards compatibility.

```
<?php
    // InterBase 6 TIME-type columns will be returned in
    // the form '05 hours 37 minutes'.
    ibase_timefmt("%H hours %M minutes", IBASE_TIME);
?>
```

You can also set defaults for these formats with PHP configuration directives `ibase.timestampformat`, `ibase.dateformat` and `ibase.timeformat`.

Note: *columntype* was added in PHP 4.0. It has any meaning only with InterBase version 6 and higher.

Note: A backwards incompatible change happened in PHP 4.0 when PHP configuration directive `ibase.timeformat` was renamed to `ibase.timestampformat` and directives `ibase.dateformat` and `ibase.timeformat` were added, so that the names would match better their functionality.

ibase_num_fields (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Get the number of fields in a result set

```
int ibase_num_fields (int result_id)
```

Returns an integer containing the number of fields in a result set.

```
<?php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);

    if (ibase_num_fields($sth) > 0) {
        while ($row = ibase_fetch_object ($sth)) {
            print $row->email . "\n";
        }
    } else {
        die ("No Results were found for your query");
    }

    ibase_close ($dbh);
?>
```

See also: **ibase_field_info()**.

Note: **ibase_num_fields()** is currently not functional in PHP 4.

ibase_errmsg (PHP 3>= 3.0.7, PHP 4 >= 4.0RC1)

Returns error messages

```
string ibase_errmsg (void )
```

Returns a string containing an error message.

XXXIII. Ingres II functions

These functions allow you to access Ingres II database servers.

In order to have these functions available, you must compile php with Ingres support by using the `-with-ingres` option. You need the Open API library and header files included with Ingres II. If the `II_SYSTEM` environment variable isn't correctly set you may have to use `-with-ingres=DIR` to specify your Ingres installation directory.

When using this extension with Apache, if Apache does not start and complains with "PHP Fatal error: Unable to start `ingres_ii` module in Unknown on line 0" then make sure the environment variable `II_SYSTEM` is correctly set. Adding "`export II_SYSTEM="/home/ingres/II"`" in the script that starts Apache, just before launching `httpd`, should be fine.

Note: If you already used PHP extensions to access other database servers, note that Ingres doesn't allow concurrent queries and/or transaction over one connection, thus you won't find any result or transaction handle in this extension. The result of a query must be treated before sending another query, and a transaction must be committed or rolled back before opening another transaction (which is automatically done when sending the first query).

ingres_connect (PHP 4 >= 4.0.2)

Open a connection to an Ingres II database.

```
resource ingres_connect ([string database [, string username [, string password]]])
```

Returns a Ingres II link resource on success, or false on failure.

ingres_connect() opens a connection with the Ingres database designated by *database*, which follows the syntax [*node_id*:::]*dbname[/svr_class]*.

If some parameters are missing, **ingres_connect()** uses the values in *php.ini* for *ingres.default_database*, *ingres.default_user* and *ingres.default_password*.

The connection is closed when the script ends or when **ingres_close()** is called on this link.

All the other *ingres* functions use the last opened link as a default, so you need to store the returned value only if you use more than one link at a time.

Example 1. **ingres_connect()** example

```
<?php
    $link = ingres_connect ("mydb", "user", "pass")
        or die ("Could not connect");
    print ("Connected successfully");
    ingres_close ($link);
?>
```

Example 2. **ingres_connect()** example using default link

```
<?php
    ingres_connect ("mydb", "user", "pass")
        or die ("Could not connect");
    print ("Connected successfully");
    ingres_close ();
?>
```

See also **ingres_pconnect()**, and **ingres_close()**.

ingres_pconnect (PHP 4 >= 4.0.2)

Open a persistent connection to an Ingres II database.

```
resource ingres_pconnect ([string database [, string username [, string password]]])
```

Returns a Ingres II link resource on success, or false on failure.

See **ingres_connect()** for parameters details and examples. There are only 2 differences between **ingres_pconnect()** and **ingres_connect()** : First, when connecting, the function will first try to find a (persistent) link that's already opened with the same parameters. If one is found, an identifier for it will be returned instead of opening a new connection. Second, the connection to the Ingres server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (**ingres_close()** will not close links established by **ingres_pconnect()**). This type of link is therefore called 'persistent'.

See also **ingres_connect()**, and **ingres_close()**.

ingres_close (PHP 4 >= 4.0.2)

Close an Ingres II database connection

```
bool ingres_close ([resource link])
```

Returns true on success, or false on failure.

ingres_close() closes the connection to the Ingres server that's associated with the specified link. If the *link* parameter isn't specified, the last opened link is used.

ingres_close() isn't usually necessary, as it won't close persistent connections and all non-persistent connections are automatically closed at the end of the script.

See also **ingres_connect()**, and **ingres_pconnect()**.

ingres_query (PHP 4 >= 4.0.2)

Send a SQL query to Ingres II

```
bool ingres_query (string query [, resource link])
```

Returns true on success, or false on failure.

ingres_query() sends the given *query* to the Ingres server. This query must be a valid SQL query (see the Ingres SQL reference guide)

The query becomes part of the currently open transaction. If there is no open transaction, **ingres_query()** opens a new transaction. To close the transaction, you can either call **ingres_commit()** to commit the changes made to the database or **ingres_rollback()** to cancel these changes. When the script ends, any open transaction is rolled back (by calling **ingres_rollback()**). You can also use **ingres_autocommit()** before opening a new transaction to have every SQL query immediately committed.

Some types of SQL queries can't be sent with this function :

- close (see **ingres_close()**).
- commit (see **ingres_commit()**).
- connect (see **ingres_connect()**).
- disconnect (see **ingres_close()**).
- get dbevent
- prepare to commit
- rollback (see **ingres_rollback()**).
- savepoint
- set autocommit (see **ingres_autocommit()**).
- all cursor related queries are unsupported

Example 1. **ingres_query()** example

```
<?php
    ingres_connect ($database, $user, $password);
```

```
ingres_query ("select * from table");
while ($row = ingres_fetch_row()) {
    echo $row[1];
    echo $row[2];
}
?>
```

See also **ingres_fetch_array()**, **ingres_fetch_object()**, **ingres_fetch_row()**, **ingres_commit()**, **ingres_rollback()** and **ingres_autocommit()**.

ingres_num_rows (PHP 4 >= 4.0.2)

Get the number of rows affected or returned by the last query

```
int ingres_num_rows ([resource link])
```

For delete, insert or update queries, **ingres_num_rows()** returns the number of rows affected by the query. For other queries, **ingres_num_rows()** returns the number of rows in the query's result.

Note: This function is mainly meant to get the number of rows modified in the database. If this function is called before using **ingres_fetch_array()**, **ingres_fetch_object()** or **ingres_fetch_row()** the server will delete the result's data and the script won't be able to get them.

You should instead retrieve the result's data using one of these fetch functions in a loop until it returns false, indicating that no more results are available.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_num_fields (PHP 4 >= 4.0.2)

Get the number of fields returned by the last query

```
int ingres_num_fields ([resource link])
```

ingres_num_fields() returns the number of fields in the results returned by the Ingres server after a call to **ingres_query()**

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_name (PHP 4 >= 4.0.2)

Get the name of a field in a query result.

```
string ingres_field_name (int index [, resource link])
```

ingres_field_name() returns the name of a field in a query result, or false on failure.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_type (PHP 4 >= 4.0.2)

Get the type of a field in a query result.

```
string ingres_field_type (int index [, resource link])
```

ingres_field_type() returns the type of a field in a query result, or false on failure. Examples of types returned are "IIAPI_BYTE_TYPE", "IIAPI_CHA_TYPE", "IIAPI_DTE_TYPE", "IIAPI_FLT_TYPE", "IIAPI_INT_TYPE", "IIAPI_VCH_TYPE". Some of these types can map to more than one SQL type depending on the length of the field (see **ingres_field_length()**). For example "IIAPI_FLT_TYPE" can be a float4 or a float8. For detailed information, see the Ingres/OpenAPI User Guide - Appendix C.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_nullable (PHP 4 >= 4.0.2)

Test if a field is nullable.

```
bool ingres_field_nullable (int index [, resource link])
```

ingres_field_nullable() returns true if the field can be set to the NULL value and false if it can't.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_length (PHP 4 >= 4.0.2)

Get the length of a field.

```
int ingres_field_length (int index [, resource link])
```

ingres_field_length() returns the length of a field. This is the number of bytes used by the server to store the field. For detailed information, see the Ingres/OpenAPI User Guide - Appendix C.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_precision (PHP 4 >= 4.0.2)

Get the precision of a field.

```
int ingres_field_precision (int index [, resource link])
```

ingres_field_precision() returns the precision of a field. This value is used only for decimal, float and money SQL data types. For detailed information, see the Ingres/OpenAPI User Guide - Appendix C.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_field_scale (PHP 4 >= 4.0.2)

Get the scale of a field.

```
int ingres_field_scale (int index [, resource link])
```

ingres_field_scale() returns the scale of a field. This value is used only for the decimal SQL data type. For detailed information, see the Ingres/OpenAPI User Guide - Appendix C.

index is the number of the field and must be between 1 and the value given by **ingres_num_fields()**.

See also **ingres_query()**, **ingres_fetch_array()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_fetch_array (PHP 4 >= 4.0.2)

Fetch a row of result into an array.

```
array ingres_fetch_array ([int result_type [, resource link]])
```

ingres_fetch_array() Returns an array that corresponds to the fetched row, or false if there are no more rows.

This function is an extended version of **ingres_fetch_row()**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use the numeric index of the column or make an alias for the column.

```
ingres_query(select t1.f1 as foo t2.f1 as bar from t1, t2);
$result = ingres_fetch_array();
$foo = $result["foo"];
$bar = $result["bar"];
```

result_type can be II_NUM for enumerated array, II_ASSOC for associative array, or II_BOTH (default).

Speed-wise, the function is identical to **ingres_fetch_object()**, and almost as quick as **ingres_fetch_row()** (the difference is insignificant).

Example 1. **ingres_fetch_array()** example

```
<?php
ingres_connect ($database, $user, $password);

ingres_query ("select * from table");
while ($row = ingres_fetch_array()) {
    echo $row["user_id"]; # using associative array
    echo $row["fullname"];
    echo $row[1];          # using enumerated array
    echo $row[2];
```

```
}
```

```
?>
```

See also **ingres_query()**, **ingres_num_fields()**, **ingres_field_name()**, **ingres_fetch_object()** and **ingres_fetch_row()**.

ingres_fetch_row (PHP 4 >= 4.0.2)

Fetch a row of result into an enumerated array.

```
arrayingres_fetch_row ([resource link])
```

ingres_fetch_row() returns an array that corresponds to the fetched row, or false if there are no more rows. Each result column is stored in an array offset, starting at offset 1.

Subsequent call to **ingres_fetch_row()** would return the next row in the result set, or false if there are no more rows.

Example 1. **ingres_fetch_row()** example

```
<?php
ingres_connect ($database, $user, $password);

ingres_query ("select * from table");
while ($row = ingres_fetch_row()) {
    echo $row[1];
    echo $row[2];
}
?>
```

See also **ingres_num_fields()**, **ingres_query()**, **ingres_fetch_array()** and **ingres_fetch_object()**.

ingres_fetch_object (PHP 4 >= 4.0.2)

Fetch a row of result into an object.

```
objectingres_fetch_object ([int result_type [, resource link]])
```

ingres_fetch_object() Returns an object that corresponds to the fetched row, or false if there are no more rows.

This function is similar to **ingres_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The optional argument *result_type* is a constant and can take the following values: **II_ASSOC**, **II_NUM**, and **II_BOTH**.

Speed-wise, the function is identical to **ingres_fetch_array()**, and almost as quick as **ingres_fetch_row()** (the difference is insignificant).

Example 1. **ingres_fetch_object()** example

```
<?php
```

```

ingres_connect ($database, $user, $password);
ingres_query ("select * from table");
while ($row = ingres_fetch_object()) {
    echo $row->user_id;
    echo $row->fullname;
}
?>

```

See also [ingres_query\(\)](#), [ingres_num_fields\(\)](#), [ingres_field_name\(\)](#), [ingres_fetch_array\(\)](#) and [ingres_fetch_row\(\)](#).

ingres_rollback (PHP 4 >= 4.0.2)

Roll back a transaction.

```
bool ingres_rollback ([resource link])
```

ingres_rollback() rolls back the currently open transaction, actually canceling all changes made to the database during the transaction.

This closes the transaction. A new one can be open by sending a query with [ingres_query\(\)](#).

See also [ingres_query\(\)](#), [ingres_commit\(\)](#) and [ingres_autocommit\(\)](#).

ingres_commit (PHP 4 >= 4.0.2)

Commit a transaction.

```
bool ingres_commit ([resource link])
```

ingres_commit() commits the currently open transaction, making all changes made to the database permanent.

This closes the transaction. A new one can be open by sending a query with [ingres_query\(\)](#).

You can also have the server commit automatically after every query by calling [ingres_autocommit\(\)](#) before opening the transaction.

See also [ingres_query\(\)](#), [ingres_rollback\(\)](#) and [ingres_autocommit\(\)](#).

ingres_autocommit (PHP 4 >= 4.0.2)

Switch autocommit on or off.

```
bool ingres_autocommit ([resource link])
```

ingres_autocommit() is called before opening a transaction (before the first call to [ingres_query\(\)](#) or just after a call to [ingres_rollback\(\)](#) or [ingres_autocommit\(\)](#)) to switch the "autocommit" mode of the server on or off (when the script begins the autocommit mode is off).

When the autocommit mode is on, every query is automatically committed by the server, as if [ingres_commit\(\)](#) was called after every call to [ingres_query\(\)](#).

See also [ingres_query\(\)](#), [ingres_rollback\(\)](#) and [ingres_commit\(\)](#).

XXXIV. 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 ".$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 entries 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 Ildap-3.3 package or the Netscape Directory SDK 3.0. 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.org/>)
- LDAP World (<http://elvira.innosoft.com/ldapworld>)

The Netscape SDK contains a helpful Programmer's Guide in .html format.

ldap_add (PHP 3, PHP 4)

Add entries to LDAP directory

```
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_bind (PHP 3, PHP 4)

Bind to LDAP directory

```
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 (PHP 3, PHP 4)

Close link to LDAP server

```
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_compare (PHP 4 >= 4.0.2)

Compare value of attribute found in entry specified with DN

```
int ldap_compare (int link_identifier, string dn, string attribute, string value)
```

Returns true if *value* matches otherwise returns false. Returns -1 on error.

ldap_compare() is used to compare *value* of *attribute* to value of same attribute in LDAP directory entry specified with *dn*.

The following example demonstrates how to check whether or not given password matches the one defined in DN specified entry.

Example 1. Complete example of password check

```
<?php

$ds=ldap_connect("localhost"); // assuming the LDAP server is on this host

if ($ds) {

    // bind
    if(ldap_bind($ds)) {

        // prepare data
        $dn = "cn=Matti Meikku, ou=My Unit, o=My Company, c=FI";
        $value = "secretpassword";
        $attr = "password";

        // compare value
        $r=ldap_compare($ds, $dn, $attr, $value);

        if ($r === -1) {
            echo "Error: ".ldap_error($ds);
        } elseif ($r === TRUE) {
            echo "Password correct.";
        } elseif ($r === FALSE) {
            echo "Wrong guess! Password incorrect.";
        }
    }

} else {
    echo "Unable to bind to LDAP server.";
}

ldap_close($ds);

} else {
    echo "Unable to connect to LDAP server.";
```

```
}
```

```
?>
```

Note: **ldap_compare()** can NOT be used to compare BINARY values!

Note: This function was added in 4.0.2.

ldap_connect (PHP 3, PHP 4)

Connect to an LDAP server

```
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.

If you are using OpenLDAP 2.x.x you can specify a URL instead of the hostname. To use LDAP with SSL, compile OpenLDAP 2.x.x with SSL support, configure PHP with SSL, and use ldaps://hostname/ as host parameter. The port parameter is not used when using URLs. URL and SSL support were added in 4.0.4.

ldap_count_entries (PHP 3, PHP 4)

Count the number of entries in a search

```
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 (PHP 3, PHP 4)

Delete an entry from a directory

```
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 (PHP 3, PHP 4)

Convert DN to User Friendly Naming format

```
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_err2str (PHP 3>= 3.0.13, PHP 4 >= 4.0RC2)

Convert LDAP error number into string error message

```
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 $i: %s<br>\n", ldap_err2str($i));
}
?>
```

ldap_errno (PHP 3>= 3.0.12, PHP 4 >= 4.0RC2)

Return the LDAP error number of the last LDAP command

```
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 (PHP 3>= 3.0.12, PHP 4 >= 4.0RC2)

Return the LDAP error message of the last LDAP command

```
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()**.

ldap_explode_dn (PHP 3, PHP 4)

Splits DN into its component parts

```
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_attrib* 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_attrib* to 0 and to get only values set it to 1.

ldap_first_attribute (PHP 3, PHP 4)

Return first attribute

```
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 (PHP 3, PHP 4)

Return first result id

```
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 (PHP 3, PHP 4)

Free result memory

```
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 (PHP 3, PHP 4)

Get attributes from a search result entry

```
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 (PHP 3, PHP 4)

Get the DN of a result entry

```
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 (PHP 3, PHP 4)

Get all result entries

```
array ldap_get_entries (int link_identifier, int result_identifier)
```

Returns a complete result information in a multi-dimenasional 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 ith entry in the result

return_value[i]["count"] = number of attributes in ith entry

`return_value[i][j]` = jth attribute in the ith entry in the result
`return_value[i]["attribute"]["count"]` = number of values for attribute in ith entry
`return_value[i]["attribute"][j]` = jth value of attribute in ith entry

see also **ldap_first_entry()** and **ldap_next_entry()**

ldap_get_option (unknown)

Get the current value for given option

```
boolean ldap_get_option (int link_identifier, int option, mixed retval)
```

Sets `retval` to the value of the specified option, and returns true on success and false on error.

The parameter `option` can be one of: LDAP_OPT_DEREF, LDAP_OPT_SIZELIMIT, LDAP_OPT_TIMELIMIT, LDAP_OPT_PROTOCOL_VERSION, LDAP_OPT_ERROR_NUMBER, LDAP_OPT_REFERRALS, LDAP_OPT_RESTART, LDAP_OPT_HOST_NAME, LDAP_OPT_ERROR_STRING, LDAP_OPT_MATCHED_DN. These are described in [\(http://www.openldap.org-devel/cvsweb.cgi/~checkout~/doc/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt\)](http://www.openldap.org-devel/cvsweb.cgi/~checkout~/doc/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt)

This function is only available when using OpenLDAP 2.x.x OR Netscape Directory SDK x.x, and was added in PHP 4.0.4

Example 1. Check protocol version

```
// $ds is a valid link identifier for a directory server
if (ldap_get_option($ds, LDAP_OPT_PROTOCOL_VERSION, $version))
    echo "Using protocol version $version";
else
    echo "Unable to determine protocol version";
```

See also **ldap_set_option()**.

ldap_get_values (PHP 3, PHP 4)

Get all values from a result entry

```
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.

Your 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 (PHP 3>= 3.0.13, PHP 4 >= 4.0RC2)

Get all binary values from a result entry

```
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.

Note: This function was added in 4.0.

ldap_list (PHP 3, PHP 4)

Single-level search

```
int ldap_list ( int link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]])
```

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 5 optional parameters. See **ldap_search()** notes.

Note: These optional parameters were added in 4.0.2: *attrsonly*, *sizelimit*, *timelimit*, *deref*.

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 (PHP 3, PHP 4)

Modify an LDAP entry

```
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_mod_add (PHP 3>= 3.0.8, PHP 4)

Add attribute values to current attributes

```
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 (PHP 3>= 3.0.8, PHP 4)

Delete attribute values from current attributes

```
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 (PHP 3>= 3.0.8, PHP 4)

Replace attribute values with new ones

```
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_next_attribute (PHP 3, PHP 4)

Get the next attribute in result

```
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 (PHP 3, PHP 4)

Get next result entry

```
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 (PHP 3, PHP 4)

Read an entry

```
int ldap_read (int link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]])
```

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 5 optional parameters. See **ldap_search()** notes.

Note: These optional parameters were added in 4.0.2: *attrsonly*, *sizelimit*, *timelimit*, *deref*.

ldap_search (PHP 3, PHP 4)

Search LDAP tree

```
int ldap_search (int link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]])
```

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. This occurs also if the sixth parameter *sizelimit* has been used to limit the count of fetched entries.

The fifth parameter *attrsonly* should be set to 1 if only attribute types are wanted. If set to 0 both attributes types and attribute values are fetched which is the default behaviour.

With the sixth parameter *sizelimit* it is possible to limit the count of entries fetched. Setting this to 0 means no limit. NOTE: This parameter can NOT override server-side preset sizelimit. You can set it lower though.

The seventh parameter *timelimit* sets the number of seconds how long is spend on the search. Setting this to 0 means no limit. NOTE: This parameter can NOT override server-side preset timelimit. You can set it lower though.

The eighth parameter *deref* specifies how aliases should be handled during the search. It can be one of the following:

- LDAP_DEREF_NEVER - (default) aliases are never dereferenced.
- LDAP_DEREF_SEARCHING - aliases should be dereferenced during the search but not when locating the base object of the search.
- LDAP_DEREF_FINDING - aliases should be dereferenced when locating the base object but not during the search.
- LDAP_DEREF_ALWAYS - aliases should be dereferenced always.

These optional parameters were added in 4.0.2: *attrsonly*, *sizelimit*, *timelimit*, *deref*.

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/docs/manuals/directory/41/ag/find.htm>) 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_set_option (unknown)

Set the value of the given option

```
boolean ldap_set_option (int link_identifier, int option, mixed newval)
```

Sets the value of the specified option to be *newval*, and returns true on success and false on error.

The parameter *option* can be one of: LDAP_OPT_DEREF, LDAP_OPT_SIZELIMIT, LDAP_OPT_TIMELIMIT, LDAP_OPT_PROTOCOL_VERSION, LDAP_OPT_ERROR_NUMBER, LDAP_OPT_REFERRALS, LDAP_OPT_RESTART, LDAP_OPT_HOST_NAME, LDAP_OPT_ERROR_STRING, LDAP_OPT_MATCHED_DN, LDAP_OPT_SERVER_CONTROLS, LDAP_OPT_CLIENT_CONTROLS. Here's a brief description, see [draft-ietf-ldapext-ldap-c-api-xx.txt](http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-c-api-xx.txt)

(<http://www.openldap.org-devel/cvsweb.cgi/~checkout~/doc/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt>) for details.

The options LDAP_OPT_DEREF, LDAP_OPT_SIZELIMIT, LDAP_OPT_TIMELIMIT, LDAP_OPT_PROTOCOL_VERSION and LDAP_OPT_ERROR_NUMBER have integer value, LDAP_OPT_REFERRALS and LDAP_OPT_RESTART have boolean value, and the options LDAP_OPT_HOST_NAME, LDAP_OPT_ERROR_STRING and LDAP_OPT_MATCHED_DN have string value. The first example illustrates their use. The options LDAP_OPT_SERVER_CONTROLS and LDAP_OPT_CLIENT_CONTROLS require a list of controls, this means that the value must be an array of controls. A control consists of an *oid* identifying the control, an optional *value*, and an optional flag for *criticality*. In PHP a control is given by an array containing an element with the key *oid* and string value, and two optional elements. The optional elements are key *value* with string value and key *iscritical* with boolean value. *iscritical* defaults to *FALSE* if not supplied. See also the second example below.

This function is only available when using OpenLDAP 2.x.x OR Netscape Directory SDK x.x, and was added in PHP 4.0.4

Example 1. Set protocol version

```
// $ds is a valid link identifier for a directory server
```

```
if (ldap_set_option($ds, LDAP_OPT_PROTOCOL_VERSION, 3))
    echo "Using LDAPv3";
else
    echo "Failed to set protocol version to 3";
```

Example 2. Set server controls

```
// $ds is a valid link identifier for a directory server
// control with no value
$ctrl1 = array("oid" => "1.2.752.58.10.1", "iscritical" => TRUE);
// iscritical defaults to FALSE
$ctrl2 = array("oid" => "1.2.752.58.1.10", "value" => "magic");
// try to set both controls
if (!ldap_set_option($ds, LDAP_OPT_SERVER_CONTROLS, array($ctrl1, $ctrl2)))
    echo "Failed to set server controls";
```

See also **ldap_get_option()**.

ldap_unbind (PHP 3, PHP 4)

Unbind from LDAP directory

```
int ldap_unbind (int link_identifier)
```

Returns true on success and false on error.

ldap_unbind() function unbinds from the LDAP directory.

XXXV. Mail functions

The **mail()** function allows you to send mail.

mail (PHP 3, PHP 4)

send mail

```
bool mail (string to, string subject, string message [, string additional_headers])
```

Mail() automatically mails the message specified in *message* to the receiver specified in *to*. Multiple recipients can be specified by putting a comma between each address in *to*.

Example 1. Sending mail.

```
mail("rasmus@lerdorf.on.ca", "My Subject", "Line 1\nLine 2\nLine 3");
```

If a fourth string argument is passed, this string is inserted at the end of the header. This is typically used to add extra headers. Multiple extra headers are separated with a newline.

Example 2. Sending mail with extra headers.

```
mail("nobody@aol.com", "the subject", $message,
    "From: webmaster@$SERVER_NAME\nReply-To: webmaster@$SERVER_NAME\nX-
Mailer: PHP/" . phpversion());
```

You can also use fairly simple string building techniques to build complex email messages.

Example 3. Sending complex email.

```
/* recipients */
$recipient .= "Mary <mary@u.college.edu>" . " "; //note the comma
$recipient .= "Kelly <kelly@u.college.edu>" . " ";
$recipient .= "ronabop@php.net";

/* subject */
$subject = "Birthday Reminders for August";

/* message */
$message .= "The following email includes a formatted ASCII table\n";
$message .= "Day \t\Month \t\Year\n";
$message .= "3rd \t\Aug \t\t1970\n";
$message .= "17rd\t\Aug \t\t1973\n";

/* you can add a stock signature */
$message .= "-\r\n"; //Signature delimiter
$message .= "Birthday reminder copylefted by public domain";

/* additional header pieces for errors, From cc's, bcc's, etc */

$headers .= "From: Birthday Reminder <birthday@php.net>\n";
$headers .= "X-Sender: <birthday@php.net>\n";
$headers .= "X-Mailer: PHP\n"; // mailer
$headers .= "X-Priority: 1\n"; // Urgent message!
$headers .= "Return-Path: <birthday@php.net>\n"; // Return path for errors

/* If you want to send html mail, uncomment the following line */
// $headers .= "Content-Type: text/html; charset=iso-8859-1\n"; // Mime type

$headers .= "cc:birthdayarchive@php.net\n"; // CC to
```

```
$headers .= "bcc:birthdaycheck@php.net, birthdaygifts@php.net\n"; // BCCs to  
/* and now mail it */  
mail($recipient, $subject, $message, $headers);
```

ezmlm_hash (PHP 3>= 3.0.17, PHP 4 >= 4.0.2)

Calculate the hash value needed by EZMLM

```
int ezmlm_hash (string addr)
```

ezmlm_hash() calculates the hash value needed when keeping EZMLM mailing lists in a MySQL database.

Example 1. Calculating the hash and subscribing a user

```
$user = "kris@koehtopp.de";  
$hash = ezmlm_hash ($user);  
$query = sprintf ("INSERT INTO sample VALUES (%s, '%s')", $hash, $user);  
$db->query($query); // using PHPLIB db interface
```

XXXVI. Mathematical Functions

Introduction

These math functions will only handle values within the range of the long and double types on your computer. If you need to handle bigger numbers, take a look at the [arbitrary precision math functions](#).

Math constants

The following values are defined as constants in PHP by the math extension:

Table 1. Math constants

Constant	Value	Description
M_PI	3.14159265358979323846	Pi
M_E	2.7182818284590452354	e
M_LOG2E	1.4426950408889634074	log_2 e
M_LOG10E	0.43429448190325182765	log_10 e
M_LN2	0.69314718055994530942	log_e 2
M_LN10	2.30258509299404568402	log_e 10
M_PI_2	1.57079632679489661923	pi/2
M_PI_4	0.78539816339744830962	pi/4
M_1_PI	0.31830988618379067154	1/pi
M_2_PI	0.63661977236758134308	2/pi
M_SQRTPI	1.77245385090551602729	sqrt(pi) [4.0.2]
M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT3	1.73205080756887729352	sqrt(3) [4.0.2]
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)
M_LNPI	1.14472988584940017414	log_e(pi) [4.0.2]
M_EULER	0.57721566490153286061	Euler constant [4.0.2]

Only M_PI is available in PHP versions up to and including PHP4RC1. All other constants are available starting with PHP 4.0. Constants labelled [4.0.2] were added in PHP 4.0.2.

abs (PHP 3, PHP 4)

Absolute value

```
mixed abs (mixed number)
```

Returns the absolute value of number. If the argument number is float, return type is also float, otherwise it is int.

acos (PHP 3, PHP 4)

Arc cosine

```
float acos (float arg)
```

Returns the arc cosine of arg in radians.

See also **asin()** and **atan()**.

asin (PHP 3, PHP 4)

Arc sine

```
float asin (float arg)
```

Returns the arc sine of arg in radians.

See also **acos()** and **atan()**.

atan (PHP 3, PHP 4)

Arc tangent

```
float atan (float arg)
```

Returns the arc tangent of arg in radians.

See also **asin()** and **acos()**.

atan2 (PHP 3>= 3.0.5, PHP 4)

arc tangent of two variables

```
float atan2 (float y, float x)
```

This function calculates the arc tangent of the two variables *x* and *y*. It is similar to calculating the arc tangent of *y / x*, except that the signs of both arguments are used to determine the quadrant of the result.

The function returns the result in radians, which is between -PI and PI (inclusive).

See also **acos()** and **atan()**.

base_convert (PHP 3>= 3.0.6, PHP 4)

Convert a number between arbitrary bases

```
string base_convert (string number, int frombase, int tobase)
```

Returns a string containing *number* represented in base *tobase*. The base in which *number* is given is specified in *frombase*. Both *frombase* and *tobase* have to be between 2 and 36, inclusive. Digits in numbers with a base higher than 10 will be represented with the letters a-z, with a meaning 10, b meaning 11 and z meaning 35.

Example 1. Base_convert()

```
$binary = base_convert ($hexadecimal, 16, 2);
```

bindec (PHP 3, PHP 4)

Binary to decimal

```
int bindec (string binary_string)
```

Returns the decimal equivalent of the binary number represented by the *binary_string* argument.

Octdec converts a binary number to a decimal number. The largest number that can be converted is 31 bits of 1's or 2147483647 in decimal.

See also the **decbin()** function.

ceil (PHP 3, PHP 4)

Round fractions up

```
int ceil (float number)
```

Returns the next highest integer value from *number*. Using **ceil()** on integers is absolutely a waste of time.

```
$x = ceil(4.25);
// which would make $x=5
```

NOTE: PHP/FI 2's **ceil()** returned a float. Use: `$new = (double)ceil($number);` to get the old behaviour.

See also **floor()** and **round()**.

COS (PHP 3, PHP 4)

Cosine

```
float cos (float arg)
```

Returns the cosine of arg in radians.

See also [sin\(\)](#) and [tan\(\)](#).

decbin (PHP 3, PHP 4)

Decimal to binary

```
string decbin (int number)
```

Returns a string containing a binary representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to a string of 31 1's.

See also the [bindec\(\)](#) function.

dechex (PHP 3, PHP 4)

Decimal to hexadecimal

```
string dechex (int number)
```

Returns a string containing a hexadecimal representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to "7fffffff".

See also the [hexdec\(\)](#) function.

decoct (PHP 3, PHP 4)

Decimal to octal

```
string decoct (int number)
```

Returns a string containing an octal representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to "17777777777".

See also [octdec\(\)](#).

deg2rad (PHP 3>= 3.0.4, PHP 4)

Converts the number in degrees to the radian equivalent

```
double deg2rad (double number)
```

This function converts *number* from degrees to the radian equivalent.

See also [rad2deg\(\)](#).

exp (PHP 3, PHP 4)

e to the power of ...

```
float exp (float arg)
```

Returns e raised to the power of *arg*.

See also **pow()**.

floor (PHP 3, PHP 4)

Round fractions down

```
int floor (float number)
```

Returns the next lowest integer value from *number*. Using **floor()** on integers is absolutely a waste of time.

NOTE: PHP/FI 2's **floor()** returned a float. Use: \$new = (double)floor(\$number); to get the old behaviour.

See also **ceil()** and **round()**.

getrandmax (PHP 3, PHP 4)

Show largest possible random value

```
int getrandmax (void)
```

Returns the maximum value that can be returned by a call to **rand()**.

See also **rand()**, **srand()**, **mt_rand()**, **mt_srand()**, and **mt_getrandmax()**.

hexdec (PHP 3, PHP 4)

Hexadecimal to decimal

```
int hexdec (string hex_string)
```

Returns the decimal equivalent of the hexadecimal number represented by the *hex_string* argument. HexDec converts a hexadecimal string to a decimal number. The largest number that can be converted is 7fffffff or 2147483647 in decimal.

See also the **dechex()** function.

lcg_value (PHP 4 >= 4.0b4)

Combined linear congruential generator

```
double lcg_value(void);
```

lcg_value() returns a pseudo random number in the range of (0, 1). The function combines two CGs with periods of 2^31 - 85 and 2^31 - 249. The period of this function is equal to the product of both primes.

log (PHP 3, PHP 4)

Natural logarithm

```
float log (float arg)
```

Returns the natural logarithm of *arg*.

log10 (PHP 3, PHP 4)

Base-10 logarithm

```
float log10 (float arg)
```

Returns the base-10 logarithm of *arg*.

max (PHP 3, PHP 4)

Find highest value

```
mixed max (mixed arg1, mixed arg2, mixed argn)
```

max() returns the numerically highest of the parameter values.

If the first parameter is an array, **max()** returns the highest value in that array. If the first parameter is an integer, string or double, you need at least two parameters and **max()** returns the biggest of these values. You can compare an unlimited number of values.

If one or more of the values is a double, all the values will be treated as doubles, and a double is returned. If none of the values is a double, all of them will be treated as integers, and an integer is returned.

min (PHP 3, PHP 4)

Find lowest value

```
mixed min (mixed arg1, mixed arg2, mixed argn)
```

Min() returns the numerically lowest of the parameter values.

If the first parameter is an array, **min()** returns the lowest value in that array. If the first parameter is an integer, string or double, you need at least two parameters and **min()** returns the lowest of these values. You can compare an unlimited number of values.

If one or more of the values is a double, all the values will be treated as doubles, and a double is returned. If none of the values is a double, all of them will be treated as integers, and an integer is returned.

mt_rand (PHP 3>= 3.0.6, PHP 4)

Generate a better random value

```
int mt_rand ([int min [, int max]])
```

Many random number generators of older libcs have dubious or unknown characteristics and are slow. By default, PHP uses the libc random number generator with the **rand()** function. **mt_rand()** function is a drop-in replacement for this. It uses a random number generator with known characteristics, the Mersenne Twister, which will produce random numbers that should be suitable for seeding some kinds of cryptography (see the home pages for details) and is four times faster than what the average libc provides. The Homepage of the Mersenne Twister can be found at <http://www.math.keio.ac.jp/~matumoto/emt.html>, and an optimized version of the MT source is available from <http://www.scp.syr.edu/~marc/hawk/twister.html> (<http://www.scp.syr.edu/~marc/hawk/twister.html>).

If called without the optional *min*, *max* arguments **mt_rand()** returns a pseudo-random value between 0 and **RAND_MAX**. If you want a random number between 5 and 15 (inclusive), for example, use **mt_rand (5, 15)**.

Remember to seed the random number generator before use with **mt_srand()**.

Note: In versions before 3.0.7 the meaning of *max* was *range*. To get the same results in these versions the short example should be **mt_rand (5, 11)** to get a random number between 5 and 15.

See also **mt_srand()**, **mt_getrandmax()**, **srand()**, **rand()** and **getrandmax()**.

mt_srand (PHP 3>= 3.0.6, PHP 4)

Seed the better random number generator

```
void mt_srand (int seed)
```

Seeds the random number generator with *seed*.

```
// seed with microseconds since last "whole" second
mt_srand ((double) microtime() * 1000000);
$randval = mt_rand();
```

See also **mt_rand()**, **mt_getrandmax()**, **srand()**, **rand()**, and **getrandmax()**.

mt_getrandmax (PHP 3>= 3.0.6, PHP 4)

Show largest possible random value

```
int mt_getrandmax (void)
```

Returns the maximum value that can be returned by a call to **mt_rand()**.

See also **mt_rand()**, **mt_srand()** **rand()**, **srand()**, and **getrandmax()**.

number_format (PHP 3, PHP 4)

Format a number with grouped thousands

```
string number_format (float number, int decimals, string dec_point, string thousands_sep)
```

Number_format() returns a formatted version of *number*. This function accepts either one, two or four parameters (not three):

If only one parameter is given, *Number* will be formatted without decimals, but with a comma (",") between every group of thousands.

If two parameters are given, *number* will be formatted with *decimals* decimals with a dot (".") in front, and a comma (",") between every group of thousands.

If all four parameters are given, *number* will be formatted with *decimals* decimals, *dec_point* instead of a dot (".") before the decimals and *thousands_sep* instead of a comma (",") between every group of thousands.

octdec (PHP 3, PHP 4)

Octal to decimal

```
int octdec (string octal_string)
```

Returns the decimal equivalent of the octal number represented by the *octal_string* argument. OctDec converts an octal string to a decimal number. The largest number that can be converted is 1777777777 or 2147483647 in decimal.

See also **decoct()**.

pi (PHP 3, PHP 4)

Get value of pi

```
double pi (void)
```

Returns an approximation of pi.

pow (PHP 3, PHP 4)

Exponential expression

```
float pow (float base, float exp)
```

Returns base raised to the power of exp.

See also **exp()**.

rad2deg (PHP 3>= 3.0.4, PHP 4)

Converts the radian number to the equivalent number in degrees

```
double rad2deg (double number)
```

This function converts *number* from radian to degrees.

See also **deg2rad()**.

rand (PHP 3, PHP 4)

Generate a random value

```
int rand ([int min [, int max]])
```

If called without the optional *min*, *max* arguments **rand()** returns a pseudo-random value between 0 and RAND_MAX. If you want a random number between 5 and 15 (inclusive), for example, use `rand (5, 15)`.

Remember to seed the random number generator before use with **rand()**.

Note: In versions before 3.0.7 the meaning of *max* was *range*. To get the same results in these versions the short example should be `rand (5, 11)` to get a random number between 5 and 15.

See also **rand()**, **getrandmax()**, **mt_rand()**, **mt_srand()**, and **mt_getrandmax()**.

round (PHP 3, PHP 4)

Rounds a float

```
double round (double val [, int precision])
```

Returns the rounded value of *val* to specified *precision* (number of digits after the decimal point).

```
$foo = round (3.4); // $foo == 3.0
$foo = round (3.5); // $foo == 4.0
$foo = round (3.6); // $foo == 4.0

$foo = round (1.95583, 2); // $foo == 1.96
```

Note: The *precision* parameter is only available in PHP 4.

See also **ceil()** and **floor()**.

sin (PHP 3, PHP 4)

Sine

```
float sin (float arg)
```

Returns the sine of *arg* in radians.

See also **cos()** and **tan()**.

sqrt (PHP 3, PHP 4)

Square root

```
float sqrt (float arg)
```

Returns the square root of *arg*.

rand (PHP 3, PHP 4)

Seed the random number generator

```
void rand (int seed)
```

Seeds the random number generator with *seed*.

```
// seed with microseconds since last "whole" second
rand ((double) microtime() * 1000000);
$randval = rand();
```

See also **rand()**, **getrandmax()**, **mt_rand()**, **mt_srand()**, and **mt_getrandmax()**.

tan (PHP 3, PHP 4)

Tangent

```
float tan (float arg)
```

Returns the tangent of *arg* in radians.

See also **sin()** and **cos()**.

XXXVII. MCAL functions

MCAL stands for Modular Calendar Access Library.

Libmcal is a C library for accessing calendars. It's written to be very modular, with plugable drivers. MCAL is the calendar equivalent of the IMAP module for mailboxes.

With mcal support, a calendar stream can be opened much like the mailbox stream with the IMAP support. Calendars can be local file stores, remote ICAP servers, or other formats that are supported by the mcal library.

Calendar events can be pulled up, queried, and stored. There is also support for calendar triggers (alarms) and reoccurring events.

With libmcal, central calendar servers can be accessed and used, removing the need for any specific database or local file programming.

To get these functions to work, you have to compile PHP with `-with-mcal`. That requires the mcal library to be installed. Grab the latest version from <http://mcal.chek.com/> and compile and install it.

The following constants are defined when using the MCAL module: MCAL_SUNDAY, MCAL_MONDAY, MCAL_TUESDAY, MCAL_WEDNESDAY, MCAL_THURSDAY, MCAL_FRIDAY, MCAL_SATURDAY, MCAL_RECUR_NONE, MCAL_RECUR_DAILY, MCAL_RECUR_WEEKLY, MCAL_RECUR_MONTHLY_MDAY, MCAL_RECUR_MONTHLY_WDAY, MCAL_RECUR_YEARLY, MCAL_JANUARY, MCAL_FEBRUARY, MCAL_MARCH, MCAL_APRL, MCAL_MAY, MCAL_JUNE, MCAL_JULY, MCAL_AUGUST, MCAL_SEPTEMBER, MCAL_OCTOBER, MCAL_NOVEMBER, and MCAL_DECEMBER. Most of the functions use an internal event structure that is unique for each stream. This alleviates the need to pass around large objects between functions. There are convenience functions for setting, initializing, and retrieving the event structure values.

mcal_open (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Opens up an MCAL connection

```
int mcal_open (string calendar, string username, string password, int options)
```

Returns an MCAL stream on success, false on error.

mcal_open() opens up an MCAL connection to the specified *calendar* store. If the optional *options* is specified, passes the *options* to that mailbox also. The streams internal event structure is also initialized upon connection.

mcal_popen (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Opens up a persistant MCAL connection

```
int mcal_popen (string calendar, string username, string password, int options)
```

Returns an MCAL stream on success, false on error.

mcal_popen() opens up an MCAL connection to the specified *calendar* store. If the optional *options* is specified, passes the *options* to that mailbox also. The streams internal event structure is also initialized upon connection.

mcal_reopen (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Reopens an MCAL connection

```
int mcal_reopen (string calendar, int options)
```

Reopens an MCAL stream to a new calendar.

mcal_reopen() reopens an MCAL connection to the specified *calendar* store. If the optional *options* is specified, passes the *options* to that mailbox also.

mcal_close (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Close an MCAL stream

```
int mcal_close (int mcal_stream, int flags)
```

Closes the given mcal stream.

mcal_create_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Create a new MCAL calendar

```
string mcal_create_calendar (int stream, string calendar)
```

Creates a new calendar named *calendar*.

mcal_rename_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Rename an MCAL calendar

```
string mcal_rename_calendar (int stream, string old_name, string new_name)
```

Renames the calendar *old_name* to *new_name*.

mcal_delete_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Delete an MCAL calendar

```
string mcal_delete_calendar (int stream, string calendar)
```

Deletes the calendar named *calendar*.

mcal_fetch_event (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Fetches an event from the calendar stream

```
object mcal_fetch_event (int mcal_stream, int event_id [, int options])
```

mcal_fetch_event() fetches an event from the calendar stream specified by *id*.

Returns an event object consisting of:

- int id - ID of that event.
- int public - TRUE if the event is public, FALSE if it is private.
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - number of minutes before the event to send an alarm/reminder.
- object start - Object containing a datetime entry.
- object end - Object containing a datetime entry.
- int recur_type - recurrence type
- int recur_interval - recurrence interval
- datetime recur_enddate - recurrence end date
- int recur_data - recurrence data

All datetime entries consist of an object that contains:

- int year - year
- int month - month

- int mday - day of month
- int hour - hour
- int min - minutes
- int sec - seconds
- int alarm - minutes before event to send an alarm

mcal_list_events (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return a list of IDs for a date or a range of dates.

```
array mcal_list_events (int mcal_stream, objectbegin_date [, object end_date])
```

Returns an array of ID's that are between the start and end dates, or if just a stream is given, uses the start and end dates in the global event structure.

mcal_list_events() function takes in an beginning date and an optional end date for a calendar stream. An array of event id's that are between the given dates or the internal event dates are returned.

mcal_append_event (PHP 4 >= 4.0RC1)

Store a new event into an MCAL calendar

```
int mcal_append_event (int mcal_stream)
```

mcal_append_event() Stores the global event into an MCAL calendar for the given stream.

Returns the id of the newly inserted event.

mcal_store_event (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Modify an existing event in an MCAL calendar

```
int mcal_store_event (int mcal_stream)
```

mcal_store_event() Stores the modifications to the current global event for the given stream.

Returns true on success and false on error.

mcal_delete_event (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Delete an event from an MCAL calendar

```
int mcal_delete_event (int mcal_stream [, int event_id])
```

mcal_delete_event() deletes the calendar event specified by the *event_id*.

Returns true.

mcal_snooze (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Turn off an alarm for an event

```
int mcal_snooze (int id)
```

mcal_snooze() turns off an alarm for a calendar event specified by the id.

Returns true.

mcal_list_alarms (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return a list of events that has an alarm triggered at the given datetime

```
array mcal_list_alarms (int mcal_stream [, int begin_year [, int begin_month [, int begin_day [, int end_year [, int end_month [, int end_day]]]]]])
```

Returns an array of event ID's that has an alarm going off between the start and end dates, or if just a stream is given, uses the start and end dates in the global event structure.

mcal_list_events() function takes in an optional beginning date and an end date for a calendar stream. An array of event id's that are between the given dates or the internal event dates are returned.

mcal_event_init (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Initializes a streams global event structure

```
int mcal_event_init (int stream)
```

mcal_event_init() initializes a streams global event structure. this effectively sets all elements of the structure to 0, or the default settings.

Returns true.

mcal_event_set_category (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the category of the streams global event structure

```
int mcal_event_set_category (int stream, string category)
```

mcal_event_set_category() sets the streams global event structure's category to the given string.

Returns true.

mcal_event_set_title (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the title of the streams global event structure

```
int mcal_event_set_title (int stream, string title)
```

mcal_event_set_title() sets the streams global event structure's title to the given string.

Returns true.

mcal_event_set_description (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the description of the streams global event structure

```
int mcal_event_set_description (int stream, string description)
```

mcal_event_set_description() sets the streams global event structure's description to the given string.

Returns true.

mcal_event_set_start (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the start date and time of the streams global event structure

```
int mcal_event_set_start (int stream, int year, int month [, int day [, int hour [, int min [, int sec]]]])
```

mcal_event_set_start() sets the streams global event structure's start date and time to the given values.

Returns true.

mcal_event_set_end (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the end date and time of the streams global event structure

```
int mcal_event_set_end (int stream, int year, int month [, int day [, int hour [, int min [, int sec]]]])
```

mcal_event_set_end() sets the streams global event structure's end date and time to the given values.

Returns true.

mcal_event_set_alarm (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the alarm of the streams global event structure

```
int mcal_event_set_alarm (int stream, int alarm)
```

mcal_event_set_alarm() sets the streams global event structure's alarm to the given minutes before the event.

Returns true.

mcal_event_set_class (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the class of the streams global event structure

```
int mcal_event_set_class (int stream, int class)
```

mcal_event_set_class() sets the streams global event structure's class to the given value. The class is either 1 for public, or 0 for private.

Returns true.

mcal_is_leap_year (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns if the given year is a leap year or not

```
int mcal_is_leap_year (int year)
```

mcal_is_leap_year() returns 1 if the given year is a leap year, 0 if not.

mcal_days_in_month (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the number of days in the given month

```
int mcal_days_in_month (int month, int leap year)
```

mcal_days_in_month() Returns the number of days in the given month, taking into account if the given year is a leap year or not.

mcal_date_valid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns true if the given year, month, day is a valid date

```
int mcal_date_valid (int year, int month, int day)
```

mcal_date_valid() Returns true if the given year, month and day is a valid date, false if not.

mcal_time_valid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns true if the given year, month, day is a valid time

```
int mcal_time_valid (int hour, int minutes, int seconds)
```

mcal_time_valid() Returns true if the given hour, minutes and seconds is a valid time, false if not.

mcal_day_of_week (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the day of the week of the given date

```
int mcal_ (int year, int month, int day)
```

mcal_day_of_week() returns the day of the week of the given date.

mcal_day_of_year (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the day of the year of the given date

```
int mcal_ (int year, int month, int day)
```

mcal_day_of_year() returns the day of the year of the given date.

mcal_date_compare (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Compares two dates

```
int mcal_date_compare (int a_year, int a_month, int a_day, int b_year, int b_month, int b_day)
```

mcal_date_compare() Compares the two given dates, returns <0, 0, >0 if a<b, a==b, a>b respectively.

mcal_next_recurrence (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns the next recurrence of the event

```
int mcal_next_recurrence (int stream, int weekstart, array next)
```

mcal_next_recurrence() returns an object filled with the next date the event occurs, on or after the supplied date. Returns empty date field if event does not occur or something is invalid. Uses weekstart to determine what day is considered the beginning of the week.

mcal_event_set_recur_none (PHP 3>= 3.0.15, PHP 4 >= 4.0RC1)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_none (int stream)
```

mcal_event_set_recur_none() sets the streams global event structure to not recur (event->recur_type is set to MCAL_RECUR_NONE).

mcal_event_set_recur_daily (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_daily (int stream, int year, int month, int day, int interval)
```

mcal_event_set_recur_daily() sets the streams global event structure's recurrence to the given value to be reoccurring on a daily basis, ending at the given date.

mcal_event_set_recur_weekly (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_weekly (int stream, int year, int month, int day, int interval,
int weekdays)
```

mcal_event_set_recur_weekly() sets the streams global event structure's recurrence to the given value to be reoccurring on a weekly basis, ending at the given date.

mcal_event_set_recur_monthly_mday (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_monthly_mday (int stream, int year, int month, int day, int
interval)
```

mcal_event_set_recur_monthly_mday() sets the streams global event structure's recurrence to the given value to be reoccurring on a monthly by month day basis, ending at the given date.

mcal_event_set_recur_monthly_wday (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_monthly_wday (int stream, int year, int month, int day, int
interval)
```

mcal_event_set_recur_monthly_wday() sets the streams global event structure's recurrence to the given value to be reoccurring on a monthly by week basis, ending at the given date.

mcal_event_set_recur_yearly (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_yearly (int stream, int year, int month, int day, int interval)
```

mcal_event_set_recur_yearly() sets the streams global event structure's recurrence to the given value to be reoccurring on a yearly basis, ending at the given date.

mcal_fetch_current_stream_event (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Returns an object containing the current streams event structure

```
object mcal_fetch_current_stream_event (int stream)
```

mcal_fetch_current_stream_event() returns the current stream's event structure as an object containing:

- int id - ID of that event.
- int public - TRUE if the event is public, FALSE if it is private.
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - number of minutes before the event to send an alarm/reminder.
- object start - Object containing a datetime entry.
- object end - Object containing a datetime entry.
- int recur_type - recurrence type
- int recur_interval - recurrence interval
- datetime recur_enddate - recurrence end date
- int recur_data - recurrence data

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour
- int min - minutes
- int sec - seconds
- int alarm - minutes before event to send an alarm

mcal_event_add_attribute (PHP 3>= 3.0.15, PHP 4 >= 4.0RC1)

Adds an attribute and a value to the streams global event structure

```
void mcal_event_add_attribute (int stream, string attribute, string value)
```

mcal_event_add_attribute() adds an attribute to the stream's global event structure with the value given by "value".

mcal_expunge (unknown)

Deletes all events marked for being expunged.

```
int mcal_expunge (int stream)
```

mcal_expunge() Deletes all events which have been previously marked for deletion.

XXXVIII. Mcrypt Encryption Functions

These functions work using mcrypt (<ftp://argeas.cs-net.gr/pub/unix/mcrypt/>).

This is an interface to the mcrypt library, which supports a wide variety of block algorithms such as DES, TripleDES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFLASH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes. Additionally, it supports RC6 and IDEA which are considered "non-free".

If you linked against libmcrypt 2.4.x, the following additional block algorithms are supported: CAST, LOKI97, RIJNDAEL, SAFERPLUS, SERPENT and the following stream ciphers: ENIGMA (crypt), PANAMA, RC4 and WAKE. With libmcrypt 2.4.x another cipher mode is also available; nOFB.

To use it, download libmcrypt-x.x.tar.gz from here (<ftp://argeas.cs-net.gr/pub/unix/mcrypt/>) and follow the included installation instructions. You need to compile PHP with the `-with-mcrypt` parameter to enable this extension. Make sure you compile libmcrypt with the option `-disable-posix-threads`.

Mcrypt can be used to encrypt and decrypt using the above mentioned ciphers. If you linked against libmcrypt-2.2.x, the four important mcrypt commands (`mcrypt_cfb()`, `mcrypt_cbc()`, `mcrypt_ecb()`, and `mcrypt_ofb()`) can operate in both modes which are named MCRYPT_ENCRYPT and MCRYPT_DECRYPT, respectively.

Example 1. Encrypt an input value with TripleDES under 2.2.x in ECB mode

```
<?php
$key = "this is a very secret key";
$input = "Let us meet at 9 o'clock at the secret place.";

$encrypted_data = mcrypt_ecb (MCRYPT_3DES, $key, $input, MCRYPT_ENCRYPT);
?>
```

This example will give you the encrypted data as a string in `$encrypted_data`.

If you linked against libmcrypt 2.4.x, these functions are still available, but it is recommended that you use the advanced functions.

Example 2. Encrypt an input value with TripleDES under 2.4.x in ECB mode

```
<?php
$key = "this is a very secret key";
$input = "Let us meet at 9 o'clock at the secret place.";

$td = mcrypt_module_open (MCRYPT_TripleDES, "", MCRYPT_MODE_ECB, "");
$iv = mcrypt_create_iv (mcrypt_enc_get_iv_size ($td), MCRYPT_RAND);
mcrypt_generic_init ($td, $key, $iv);
$encrypted_data = mcrypt_generic ($td, $input);
mcrypt_generic_end ($td);
?>
```

This example will give you the encrypted data as a string in `$encrypted_data`.

Mcrypt can operate in four block cipher modes (CBC, OFB, CFB, and ECB). If linked against libmcrypt-2.4.x mcrypt can also operate in the block cipher mode nOFB and in STREAM mode. Then there are also constants in the form

MCRYPT_MODE_mode for use with several functions. We will outline the normal use for each of these modes. For a more complete reference and discussion see Applied Cryptography by Schneier (ISBN 0-471-11709-9).

- ECB (electronic codebook) is suitable for random data, such as encrypting other keys. Since data there is short and random, the disadvantages of ECB have a favorable negative effect.
- CBC (cipher block chaining) is especially suitable for encrypting files where the security is increased over ECB significantly.
- CFB (cipher feedback) is the best mode for encrypting byte streams where single bytes must be encrypted.
- OFB (output feedback, in 8bit) is comparable to CFB, but can be used in applications where error propagation cannot be tolerated. It's insecure (because it operates in 8bit mode) so it is not recommended to use it.
- nOFB (output feedback, in nbit) is comparable to OFB, but more secure because it operates on the block size of the algorithm.
- STREAM is an extra mode to include some stream algorithms like WAKE or RC4.

PHP does not support encrypting/decrypting bit streams currently. As of now, PHP only supports handling of strings.

For a complete list of supported ciphers, see the defines at the end of `mcrypt.h`. The general rule with the mcrypt-2.2.x API is that you can access the cipher from PHP with MCRYPT_ciphername. With the mcrypt-2.4.x API these constants also work, but it is possible to specify the name of the cipher as a string with a call to **mcrypt_module_open()**.

Here is a short list of ciphers which are currently supported by the mcrypt extension. If a cipher is not listed here, but is

listed by mcrypt as supported, you can safely assume that this documentation is outdated.

- MCRYPT_3DES
- MCRYPT_ARCFOUR_IV (libmcrypt 2.4.x only)
- MCRYPT_ARCFOUR (libmcrypt 2.4.x only)
- MCRYPT_BLOWFISH
- MCRYPT_CAST_128
- MCRYPT_CAST_256
- MCRYPT_CRYPT
- MCRYPT_DES
- MCRYPT_DES_COMPAT (libmcrypt 2.2.x only)
- MCRYPT_ENIGMA (libmcrypt 2.4.x only, alias for MCRYPT_CRYPT)
- MCRYPT_GOST
- MCRYPT_IDEA (non-free)
- MCRYPT_LOKI97 (libmcrypt 2.4.x only)
- MCRYPT_MARS (libmcrypt 2.4.x only, non-free)
- MCRYPT_PANAMA (libmcrypt 2.4.x only)
- MCRYPT_RIJNDAEL_128 (libmcrypt 2.4.x only)
- MCRYPT_RIJNDAEL_192 (libmcrypt 2.4.x only)
- MCRYPT_RIJNDAEL_256 (libmcrypt 2.4.x only)
- MCRYPT_RC2
- MCRYPT_RC4 (libmcrypt 2.2.x only)
- MCRYPT_RC6 (libmcrypt 2.4.x only)
- MCRYPT_RC6_128 (libmcrypt 2.2.x only)
- MCRYPT_RC6_192 (libmcrypt 2.2.x only)
- MCRYPT_RC6_256 (libmcrypt 2.2.x only)
- MCRYPT_SAFER64
- MCRYPT_SAFER128
- MCRYPT_SAFERPLUS (libmcrypt 2.4.x only)
- MCRYPT_SERPENT (libmcrypt 2.4.x only)
- MCRYPT_SERPENT_128 (libmcrypt 2.2.x only)
- MCRYPT_SERPENT_192 (libmcrypt 2.2.x only)
- MCRYPT_SERPENT_256 (libmcrypt 2.2.x only)
- MCRYPT_SKIPJACK (libmcrypt 2.4.x only)
- MCRYPT_TEAN (libmcrypt 2.2.x only)
- MCRYPT_THREEWAY
- MCRYPT_TRIPLEDES (libmcrypt 2.4.x only)
- MCRYPT_TWOFISH (for older mcrypt 2.x versions, or mcrypt 2.4.x)
- MCRYPT_TWOFISH128 (TWOFISHxxx are available in newer 2.x versions, but not in the 2.4.x versions)
- MCRYPT_TWOFISH192
- MCRYPT_TWOFISH256
- MCRYPT_WAKE (libmcrypt 2.4.x only)
- MCRYPT_XTEA (libmcrypt 2.4.x only)

function. The IV must be unique and must be the same when decrypting/encrypting. With data which is stored encrypted, you can take the output of a function of the index under which the data is stored (e.g. the MD5 key of the filename). Alternatively, you can transmit the IV together with the encrypted data (see chapter 9.3 of Applied Cryptography by Schneier (ISBN 0-471-11709-9) for a discussion of this topic).

mcrypt_get_cipher_name (PHP 3>= 3.0.8, PHP 4)

Get the name of the specified cipher

```
string mcrypt_get_cipher_name ( int cipher )

string mcrypt_get_cipher_name ( string cipher )
```

Mcrypt_get_cipher_name() is used to get the name of the specified cipher.

Mcrypt_get_cipher_name() takes the cipher number as an argument (libmcrypt 2.2.x) or takes the cipher name as an argument (libmcrypt 2.4.x) and returns the name of the cipher or false, if the cipher does not exist.

Example 1. Mcrypt_get_cipher_name() Example

```
<?php
$cipher = MCRYPT_TripleDES;

print mcrypt_get_cipher_name ($cipher);
?>
```

The above example will produce:

3DES

mcrypt_get_block_size (PHP 3>= 3.0.8, PHP 4)

Get the block size of the specified cipher

```
int mcrypt_get_block_size ( int cipher )
int mcrypt_get_block_size ( string cipher, string module )
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_get_block_size() is used to get the size of a block of the specified *cipher*.

Mcrypt_get_block_size() takes one or two arguments, the *cipher* and *module*, and returns the size in bytes.

See also: **mcrypt_get_key_size()**.

mcrypt_get_key_size (PHP 3>= 3.0.8, PHP 4)

Get the key size of the specified cipher

```
int mcrypt_get_key_size ( int cipher )
int mcrypt_get_key_size ( string cipher, string module )
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_get_key_size() is used to get the size of a key of the specified *cipher*.

Mcrypt_get_key_size() takes one or two arguments, the *cipher* and *module*, and returns the size in bytes.

See also: **mcrypt_get_block_size()**.

mcrypt_create_iv (PHP 3>= 3.0.8, PHP 4)

Create an initialization vector (IV) from a random source

```
string mcrypt_create_iv (int size, int source)
```

Mcrypt_create_iv() is used to create an IV.

mcrypt_create_iv() takes two arguments, *size* determines the size of the IV, *source* specifies the source of the IV.

The source can be MCRYPT_RAND (system random number generator), MCRYPT_DEV_RANDOM (read data from /dev/random) and MCRYPT_DEV_URANDOM (read data from /dev/urandom). If you use MCRYPT_RAND, make sure to call srand() before to initialize the random number generator.

Example 1. Mcrypt_create_iv() example

```
<?php
$cipher = MCRYPT_TripleDES;
$block_size = mcrypt_get_block_size ($cipher);
$iv = mcrypt_create_iv ($block_size, MCRYPT_DEV_RANDOM);
?>
```

mcrypt_cbc (PHP 3>= 3.0.8, PHP 4)

Encrypt/decrypt data in CBC mode

```
string mcrypt_cbc (int cipher, string key, string data, int mode [, string iv])

string mcrypt_cbc (string cipher, string key, string data, int mode [, string iv])
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_cbc() encrypts or decrypts (depending on *mode*) the *data* with *cipher* and *key* in CBC cipher mode and returns the resulting string.

Cipher is one of the MCRYPT_ciphername constants.

Key is the key supplied to the algorithm. It must be kept secret.

Data is the data which shall be encrypted/decrypted.

Mode is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

IV is the optional initialization vector.

See also: **mcrypt_cfb()**, **mcrypt_ecb()**, and **mcrypt_ofb()**.

mcrypt_cfb (PHP 3>= 3.0.8, PHP 4)

Encrypt/decrypt data in CFB mode

```
string mcrypt_cfb (int cipher, string key, string data, int mode, string iv)

string mcrypt_cfb (string cipher, string key, string data, int mode [, string iv])
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_cfb() encrypts or decrypts (depending on *mode*) the *data* with *cipher* and *key* in CFB cipher mode and returns the resulting string.

Cipher is one of the MCRYPT_ciphername constants.

Key is the key supplied to the algorithm. It must be kept secret.

Data is the data which shall be encrypted/decrypted.

Mode is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

IV is the initialization vector.

See also: **mcrypt_cbc()**, **mcrypt_ecb()**, and **mcrypt_ofb()**.

mcrypt_ecb (PHP 3>= 3.0.8, PHP 4)

Encrypt/decrypt data in ECB mode

```
string mcrypt_ecb (int cipher, string key, string data, int mode)

string mcrypt_ecb (string cipher, string key, string data, int mode [, string iv])
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_ecb() encrypts or decrypts (depending on *mode*) the *data* with *cipher* and *key* in ECB cipher mode and returns the resulting string.

Cipher is one of the MCRYPT_ciphername constants.

Key is the key supplied to the algorithm. It must be kept secret.

Data is the data which shall be encrypted/decrypted.

Mode is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

See also: **mcrypt_cbc()**, **mcrypt_cfb()**, and **mcrypt_ofb()**.

mcrypt_ofb (PHP 3>= 3.0.8, PHP 4)

Encrypt/decrypt data in OFB mode

```
string mcrypt_ofb (int cipher, string key, string data, int mode, string iv)

string mcrypt_ofb (string cipher, string key, string data, int mode [, string iv])
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_ofb() encrypts or decrypts (depending on *mode*) the *data* with *cipher* and *key* in OFB cipher mode and returns the resulting string.

Cipher is one of the MCRYPT_ciphername constants.

Key is the key supplied to the algorithm. It must be kept secret.

Data is the data which shall be encrypted/decrypted.

Mode is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

IV is the initialization vector.

See also: **mcrypt_cbc()**, **mcrypt_cfb()**, and **mcrypt_ecb()**.

mcrypt_list_algorithms (PHP 4 >= 4.0.2)

Get an array of all supported ciphers

```
array mcrypt_list_algorithms ([string lib_dir])
```

Mcrypt_list_algorithms() is used to get an array of all supported algorithms in the

lib_dir. **Mcrypt_list_algorithms()** takes as optional parameter a directory which specifies the directory where all algorithms are located. If not specified, the value of the mcrypt.algorithms_dir php.ini directive is used.

Example 1. Mcrypt_list_algorithms() Example

```
<?php
$algorithms = mcrypt_list_algorithms ( "/usr/local/lib/libmcrypt" );

foreach ($algorithms as $cipher) {
echo $cipher."/n";
}
?>
```

The above example will produce a list with all supported algorithms in the "/usr/local/lib/libmcrypt" directory.

mcrypt_list_modes (PHP 4 >= 4.0.2)

Get an array of all supported modes

```
array mcrypt_list_modes ([string lib_dir])
```

Mcrypt_list_modes() is used to get an array of all supported modes in the *lib_dir*.

Mcrypt_list_modes() takes as optional parameter a directory which specifies the directory where all modes are located. If not specified, the value of the mcrypt.modes_dir php.ini directive is used.

Example 1. Mcrypt_list_modes() Example

```
<?php
```

```
$modes = mcrypt_list_modes ();

foreach ($modes as $mode) {
echo "$mode </br>";
}
?>
```

The above example will produce a list with all supported algorithms in the default mode directory. If it is not set with the ini directive mcrypt.modes_dir, the default directory of mcrypt is used (which is /usr/local/lib/libmcrypt).

mcrypt_get_iv_size (PHP 4 >= 4.0.2)

Returns the size of the IV belonging to a specific cipher/mode combination

```
int mcrypt_get_iv_size (string cipher, string mode)
int mcrypt_get_iv_size (resource td)
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

Mcrypt_get_iv_size() returns the size of the Initialisation Vector (IV) in bytes. On error the function returns FALSE. If the IV is ignored in the specified cipher/mode combination zero is returned.

Cipher is one of the MCRYPT_ciphername constants of the name of the algorithm as string.

Mode is one of the MCRYPT_MODE_modename constants of one of "ecb", "cbc", "cfb", "ofb", "nofb" or "stream".

Td is the algorithm specified.

mcrypt_encrypt (PHP 4 >= 4.0.2)

Encrypts plaintext with given parameters

```
string mcrypt_encrypt (string cipher, string key, string data, string mode [, string iv])
```

Mcrypt_encrypt() encrypts the data and returns the encrypted data.

Cipher is one of the MCRYPT_ciphername constants of the name of the algorithm as string.

Key is the key with which the data will be encrypted. If it's smaller than the required keysize, it is padded with '\0'. It is better not to use ASCII strings for keys. It is recommended to use the mhash functions to create a key from a string.

Data is the data that will be encrypted with the given cipher and mode. If the size of the data is not n * blocksize, the data will be padded with '\0'. The returned ciphertext can be larger than the size of the data that is given by *data*.

Mode is one of the MCRYPT_MODE_modename constants of one of "ecb", "cbc", "cfb", "ofb", "nofb" or "stream".

The *IV* parameter is used for the initialisation in CBC, CFB, OFB modes, and in some algorithms in STREAM mode. If you do not supply an IV, while it is needed for an algorithm, the function issues a warning and uses an IV with all bytes set to '\0'.

Example 1. Mcrypt_encrypt() Example

```
<?php
$iv = mcrypt_create_iv (mcrypt_get_iv_size (MCRYPT_RIJNDAEL_256, MCRYPT_MODE_ECB), MCRYPT_RAND);
$key = "This is a very secret key";
```

```
$text = "Meet me at 11 o'clock behind the monument.";
echo strlen ($text)."\n";

$crypttext = mcrypt_encrypt (MCRYPT_RIJNDAEL_256, $key, $text, MCRYPT_MODE_ECB, $iv);
echo strlen ($crypttext)."\n";
?>
```

The above example will print out:

```
42
64
```

mcrypt_decrypt (PHP 4 >= 4.0.2)

Decrypts crypttext with given parameters

```
string mcrypt_decrypt (string cipher, string key, string data, string mode [, string iv])
```

Mcrypt_decrypt() decrypts the data and returns the unencrypted data.

Cipher is one of the MCRYPT_ciphername constants of the name of the algorithm as string.

Key is the key with which the data is encrypted. If it's smaller than the required keysize, it is padded with '\0'.

Data is the data that will be decrypted with the given cipher and mode. If the size of the data is not n * blocksize, the data will be padded with '\0'.

Mode is one of the MCRYPT_MODE_modename constants of one of "ecb", "cbc", "cfb", "ofb", "nofb" or "stream".

The *IV* parameter is used for the initialisation in CBC, CFB, OFB modes, and in some algorithms in STREAM mode. If you do not supply an IV, while it is needed for an algorithm, the function issues a warning and uses an IV with all bytes set to '\0'.

mcrypt_module_open (PHP 4 >= 4.0.2)

This function opens the module of the algorithm and the mode to be used

```
resource mcrypt_module_open (string algorithm, string algorithm_directory, string mode,
string mode_directory)
```

This function opens the module of the algorithm and the mode to be used. The name of the algorithm is specified in *algorithm*, eg "twofish" or is one of the MCRYPT_ciphername constants. The library is closed by calling **mcrypt_module_close()**, but there is no need to call that function if **mcrypt_generic_end()** is called. Normally it returns an encryption descriptor, or FALSE on error.

The *algorithm_directory* and *mode_directory* are used to locate the encryption modules. When you supply a directory name, it is used. When you set one of these to the empty string (""), the value set by the *mcrypt.algorithms_dir* or *mcrypt.modes_dir* ini-directive is used. When these are not set, the default directory are used that are compiled in into libmcrypt (usually /usr/local/lib/libmcrypt).

Example 1. Mcrypt_module_open() Example

```
<?php
```

```
$td = mcrypt_module_open (MCRYPT_DES, "", MCRYPT_MODE_ECB, "/usr/lib/mcrypt-modes");
?>
```

The above example will try to open the DES cipher from the default directory and the ECB mode from the directory /usr/lib/mcrypt-modes.

mcrypt_generic_init (PHP 4 >= 4.0.2)

This function initializes all buffers needed for encryption

```
int mcrypt_generic_init (resource td, string key, string iv)
```

The maximum length of the key should be the one obtained by calling **mcrypt_enc_get_key_size()** and every value smaller than this is legal. The IV should normally have the size of the algorithms block size, but you must obtain the size by calling **mcrypt_enc_get_iv_size()**. IV is ignored in ECB. IV MUST exist in CFB, CBC, STREAM, nOFB and OFB modes. It needs to be random and unique (but not secret). The same IV must be used for encryption/decryption. If you do not want to use it you should set it to zeros, but this is not recommended. The function returns (-1) on error.

You need to call this function before every **mcrypt_generic()** or **mdecrypt_generic()**.

mcrypt_generic (PHP 4 >= 4.0.2)

This function encrypts data

```
string mcrypt_generic (resource td, string data)
```

This function encrypts data. The data is padded with "\0" to make sure the length of the data is n * blocksize. This function returns the encrypted data. Note that the length of the returned string can in fact be longer then the input, due to the padding of the data.

mdecrypt_generic (PHP 4 >= 4.0.2)

This function decrypts data

```
string mdecrypt_generic (resource td, string data)
```

This function decrypts data. Note that the length of the returned string can in fact be longer then the unencrypted string, due to the padding of the data.

Example 1. Mdecrypt_generic() Example

```
<?php
$iv_size = mcrypt_enc_get_iv_size ($td);
$iv = @mcrypt_create_iv ($iv_size, MCRYPT_RAND);

if (@mcrypt_generic_init ($td, $key, $iv) != -1)
{
    $c_t = mcrypt_generic ($td, $plain_text);
    @mcrypt_generic_init ($td, $key, $iv);
    $p_t = mdecrypt_generic ($td, $c_t);
```

```

}
if (strcmp ($p_t, $plain_text, strlen($plain_text)) == 0)
    echo "ok";
else
    echo "error";
?>

```

The above example shows how to check if the data before the encryption is the same as the data after the decryption.

mcrypt_generic_end (PHP 4 >= 4.0.2)

This function terminates encryption

```
bool mcrypt_generic_end (resource td)
```

This function terminates encryption specified by the encryption descriptor (td). Actually it clears all buffers, and closes all the modules used. Returns FALSE on error, or TRUE on success.

mcrypt_enc_self_test (PHP 4 >= 4.0.2)

This function runs a self test on the opened module

```
int mcrypt_enc_self_test (resource td)
```

This function runs the self test on the algorithm specified by the descriptor td. If the self test succeeds it returns zero. In case of an error, it returns 1.

mcrypt_enc_is_block_algorithm_mode (PHP 4 >= 4.0.2)

Checks whether the encryption of the opened mode works on blocks

```
int mcrypt_enc_is_block_algorithm_mode (resource td)
```

This function returns 1 if the mode is for use with block algorithms, otherwise it returns 0. (eg. 0 for stream, and 1 for cbc, cfb, ofb).

mcrypt_enc_is_block_algorithm (PHP 4 >= 4.0.2)

Checks whether the algorithm of the opened mode is a block algorithm

```
int mcrypt_enc_is_block_algorithm (resource td)
```

This function returns 1 if the algorithm is a block algorithm, or 0 if it is a stream algorithm.

mcrypt_enc_is_block_mode (PHP 4 >= 4.0.2)

Checks whether the opened mode outputs blocks

```
int mcrypt_enc_is_block_mode (resource td)
```

This function returns 1 if the mode outputs blocks of bytes or 0 if it outputs bytes. (eg. 1 for cbc and ecb, and 0 for cfb and stream).

mcrypt_enc_get_block_size (PHP 4 >= 4.0.2)

Returns the blocksize of the opened algorithm

```
int mcrypt_enc_get_block_size (resource td)
```

This function returns the block size of the algorithm specified by the encryption descriptor td in bytes.

mcrypt_enc_get_key_size (PHP 4 >= 4.0.2)

Returns the maximum supported keysize of the opened mode

```
int mcrypt_enc_get_key_size (resource td)
```

This function returns the maximum supported key size of the algorithm specified by the encryption descriptor td in bytes.

mcrypt_enc_get_supported_key_sizes (PHP 4 >= 4.0.2)

Returns an array with the supported keysizes of the opened algorithm

```
array mcrypt_enc_get_supported_key_sizes (resource td)
```

Returns an array with the key sizes supported by the algorithm specified by the encryption descriptor. If it returns an empty array then all key sizes between 1 and **mcrypt_enc_get_key_size()** are supported by the algorithm.

mcrypt_enc_get_iv_size (PHP 4 >= 4.0.2)

Returns the size of the IV of the opened algorithm

```
int mcrypt_enc_get_iv_size (resource td)
```

This function returns the size of the iv of the algorithm specified by the encryption descriptor in bytes. If it returns '0' then the IV is ignored in the algorithm. An IV is used in cbc, cfb and ofb modes, and in some algorithms in stream mode.

mcrypt_enc_get_algorithms_name (PHP 4 >= 4.0.2)

Returns the name of the opened algorithm

```
string mcrypt_enc_get_algorithms_name (resource td)
```

This function returns the name of the algorithm.

mcrypt_enc_get_modes_name (PHP 4 >= 4.0.2)

Returns the name of the opened mode

```
string mcrypt_enc_get_modes_name (resource td)
```

This function returns the name of the mode.

mcrypt_module_self_test (PHP 4 >= 4.0.2)

This function runs a self test on the specified module

```
bool mcrypt_module_self_test (string algorithm [, string lib_dir])
```

This function runs the self test on the algorithm specified. The optional *lib_dir* parameter can contain the location of where the algorithm module is on the system.

The function returns TRUE if the self test succeeds, or FALSE when it fails.

mcrypt_module_is_block_algorithm_mode (PHP 4 >= 4.0.2)

This function returns if the the specified module is a block algorithm or not

```
bool mcrypt_module_is_block_algorithm_mode (string mode [, string lib_dir])
```

This function returns TRUE if the mode is for use with block algorithms, otherwise it returns 0. (eg. 0 for stream, and 1 for cbc, cfb, ofb). The optional *lib_dir* parameter can contain the location where the mode module is on the system.

mcrypt_module_is_block_algorithm (PHP 4 >= 4.0.2)

This function checks whether the specified algorithm is a block algorithm

```
bool mcrypt_module_is_block_algorithm (string algorithm [, string lib_dir])
```

This function returns TRUE if the specified algorithm is a block algorithm, or FALSE if it is a stream algorithm. The optional *lib_dir* parameter can contain the location where the algorithm module is on the system.

mcrypt_module_is_block_mode (PHP 4 >= 4.0.2)

This function returns if the the specified mode outputs blocks or not

```
bool mcrypt_module_is_block_mode (string mode [, string lib_dir])
```

This function returns TRUE if the mode outputs blocks of bytes or FALSE if it outputs just bytes. (eg. 1 for cbc and ecb, and 0 for cf8 and stream). The optional *lib_dir* parameter can contain the location where the mode module is on the system.

mcrypt_module_get_algo_block_size (PHP 4 >= 4.0.2)

Returns the blocksize of the specified algorithm

```
int mcrypt_module_get_algo_block_size (string algorithm [, string lib_dir])
```

This function returns the block size of the algorithm specified in bytes. The optional *lib_dir* parameter can contain the location where the mode module is on the system.

mcrypt_module_get_algo_key_size (PHP 4 >= 4.0.2)

Returns the maximum supported keysize of the opened mode

```
int mcrypt_module_get_algo_key_size (string algorithm [, string lib_dir])
```

This function returns the maximum supported key size of the algorithm specified in bytes. The optional *lib_dir* parameter can contain the location where the mode module is on the system.

mcrypt_module_get_algo_supported_key_sizes (unknown)

Returns an array with the supported keysizes of the opened algorithm

```
array mcrypt_module_enc_get_algo_supported_key_sizes (string algorithm [, string lib_dir])
```

Returns an array with the key sizes supported by the specified algorithm. If it returns an empty array then all key sizes between 1 and **mcrypt_module_get_algo_key_size()** are supported by the algorithm. The optional *lib_dir* parameter can contain the location where the mode module is on the system.

XXXIX. Mhash Functions

These functions are intended to work with mhash (<http://mhash.sourceforge.net/>).

This is an interface to the mhash library. mhash supports a wide variety of hash algorithms such as MD5, SHA1, GOST, and many others.

To use it, download the mhash distribution from its web site (<http://mhash.sourceforge.net/>) and follow the included installation instructions. You need to compile PHP with the `-with-mhash` parameter to enable this extension.

Mhash can be used to create checksums, message digests, message authentication codes, and more.

Example 1. Compute the MD5 digest and hmac and print it out as hex

```
<?php
$input = "what do ya want for nothing?";
$hash = mhash (MHASH_MD5, $input);
print "The hash is ".bin2hex ($hash)."\n<br>";
$hash = mhash (MHASH_MD5, $input, "Jefe");
print "The hmac is ".bin2hex ($hash)."\n<br>";
?>
```

This will produce:

```
The hash is d03cb659cbf9192dcd066272249f8412
The hmac is 750c783e6ab0b503eaa86e310a5db738
```

For a complete list of supported hashes, refer to the documentation of mhash. The general rule is that you can access the hash algorithm from PHP with MHASH_HASHNAME. For example, to access TIGER you use the PHP constant MHASH_TIGER.

Here is a list of hashes which are currently supported by mhash. If a hash is not listed here, but is listed by mhash as supported, you can safely assume that this documentation is outdated.

- MHASH_MD5
- MHASH_SHA1
- MHASH_HAVAL256
- MHASH_HAVAL192
- MHASH_HAVAL160
- MHASH_HAVAL128
- MHASH_RIPEMD160
- MHASH_GOST
- MHASH_TIGER
- MHASH_CRC32
- MHASH_CRC32B

mhash_get_hash_name (PHP 3>= 3.0.9, PHP 4)

Get the name of the specified hash

```
string mhash_get_hash_name (int hash)
```

Mhash_get_hash_name() is used to get the name of the specified hash.

mhash_get_hash_name() takes the hash id as an argument and returns the name of the hash or false, if the hash does not exist.

Example 1. Mhash_get_hash_name() Example

```
<?php
$hash = MHASH_MD5;

print mhash_get_hash_name ($hash);
?>
```

The above example will print out:

MD5

mhash_get_block_size (PHP 3>= 3.0.9, PHP 4)

Get the block size of the specified hash

```
int mhash_get_block_size (int hash)
```

Mhash_get_block_size() is used to get the size of a block of the specified *hash*.

Mhash_get_block_size() takes one argument, the *hash* and returns the size in bytes or false, if the *hash* does not exist.

mhash_count (PHP 3>= 3.0.9, PHP 4)

Get the highest available hash id

```
int mhash_count (void)
```

Mhash_count() returns the highest available hash id. Hashes are numbered from 0 to this hash id.

Example 1. Traversing all hashes

```
<?php
$nr = mhash_count();

for ($i = 0; $i <= $nr; $i++) {
    echo sprintf ("The blocksize of %s is %d\n",
        mhash_get_hash_name ($i),
```

```

    mhash_get_block_size ($i));
}
?>

```

mhash (PHP 3>= 3.0.9, PHP 4)

Compute hash

```
string mhash (int hash, string data, string [ key ])
```

Mhash() applies a hash function specified by *hash* to the *data* and returns the resulting hash (also called digest). If the *key* is specified it will return the resulting HMAC. HMAC is keyed hashing for message authentication, or simply a message digest that depends on the specified key. Not all algorithms supported in mhash can be used in HMAC mode. In case of an error returns false.

mhash_keygen_s2k (unknown)

Generates a key

```
string mhash_keygen_s2k (int hash, string password, string salt, int bytes)
```

Mhash_keygen_s2k() generates a key that is *bytes* long, from a user given password. This is the Salted S2K algorithm as specified in the OpenPGP document (RFC 2440). That algorithm will use the specified *hash* algorithm to create the key. The *salt* must be different and random enough for every key you generate in order to create different keys. That salt must be known when you check the keys, thus it is a good idea to append the key to it. Salt has a fixed length of 8 bytes and will be padded with zeros if you supply less bytes. Keep in mind that user supplied passwords are not really suitable to be used as keys in cryptographic algorithms, since users normally choose keys they can write on keyboard. These passwords use only 6 to 7 bits per character (or less). It is highly recommended to use some kind of transformation (like this function) to the user supplied key.

XL. Microsoft SQL Server functions

mssql_close (PHP 3, PHP 4)

Close MS SQL Server connection

```
int mssql_close ([int link_identifier])
```

Returns: true on success, false on error.

Mssql_close() closes the link to a MS SQL Server 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.

Mssql_close() will not close persistent links generated by **mssql_pconnect()**.

See also: **mssql_connect()**, **mssql_pconnect()**.

mssql_connect (PHP 3, PHP 4)

Open MS SQL server connection

```
int mssql_connect ([stringservername [, stringusername [, stringpassword]]])
```

Returns: A positive MS SQL link identifier on success, or false on error.

Mssql_connect() establishes a connection to a MS SQL server. The servername argument has to be a valid servername that is defined in the 'interfaces' file.

In case a second call is made to **mssql_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 **mssql_close()**.

See also **mssql_pconnect()**, **mssql_close()**.

mssql_data_seek (PHP 3, PHP 4)

Move internal row pointer

```
int mssql_data_seek (int result_identifier, int row_number)
```

Returns: true on success, false on failure.

Mssql_data_seek() moves the internal row pointer of the MS SQL result associated with the specified result identifier to point to the specified row number. The next call to **mssql_fetch_row()** would return that row.

See also: **mssql_data_seek()**.

mssql_fetch_array (PHP 3, PHP 4)

Fetch row as array

```
int mssql_fetch_array (int result)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

Mssql_fetch_array() is an extended version of **mssql_fetch_row()**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

An important thing to note is that using **mssql_fetch_array()** is NOT significantly slower than using **mssql_fetch_row()**, while it provides a significant added value.

For further details, also see **mssql_fetch_row()**.

mssql_fetch_field (PHP 3, PHP 4)

Get field information

```
object mssql_fetch_field (int result [, int field_offset])
```

Returns an object containing field information.

Mssql_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retrieved by **mssql_fetch_field()** is retrieved.

The properties of the object are:

- name - column name. if the column is a result of a function, this property is set to computed#N, where #N is a serial number.
- column_source - the table from which the column was taken
- max_length - maximum length of the column
- numeric - 1 if the column is numeric

See also **mssql_field_seek()**.

mssql_fetch_object (PHP 3, PHP 4)

Fetch row as object

```
int mssql_fetch_object (int result)
```

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

Mssql_fetch_object() is similar to **mssql_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

Speed-wise, the function is identical to **mssql_fetch_array()**, and almost as quick as **mssql_fetch_row()** (the difference is insignificant).

See also: **mssql_fetch-array()** and **mssql_fetch-row()**.

mssql_fetch_row (PHP 3, PHP 4)

Get row as enumerated array

```
array mssql_fetch_row (int result)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

Mssql_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 **mssql_fetch_rows()** would return the next row in the result set, or false if there are no more rows.

See also: **mssql_fetch_array()**, **mssql_fetch_object()**, **mssql_data_seek()**, **mssql_fetch_lengths()**, and **mssql_result()**.

mssql_field_length (PHP 3>= 3.0.3, PHP 4 >= 4.0b4)

Get the length of a field

```
int mssql_field_length (int result [, int offset])
```

mssql_field_name (PHP 3>= 3.0.3, PHP 4 >= 4.0b4)

Get the name of a field

```
int mssql_field_name (int result [, int offset])
```

mssql_field_seek (PHP 3, PHP 4)

Set field offset

```
int mssql_field_seek (int result, int field_offset)
```

Seeks to the specified field offset. If the next call to **mssql_fetch_field()** won't include a field offset, this field would be returned.

See also: **mssql_fetch_field()**.

mssql_field_type (PHP 3>= 3.0.3, PHP 4 >= 4.0b4)

Get the type of a field

```
string mssql_field_type (int result [, int offset])
```

mssql_free_result (PHP 3, PHP 4)

Free result memory

```
int mssql_free_result (int result)
```

mssql_free_result() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script ends. You may call **mssql_free_result()** with the result identifier as an argument and the associated result memory will be freed.

mssql_get_last_message (PHP 3, PHP 4)

Returns the last message from server (over min_message_severity?)

```
string mssql_get_last_message (void )
```

mssql_min_error_severity (PHP 3, PHP 4)

Sets the lower error severity

```
void mssql_min_error_severity (int severity)
```

mssql_min_message_severity (PHP 3, PHP 4)

Sets the lower message severity

```
void mssql_min_message_severity (int severity)
```

mssql_num_fields (PHP 3, PHP 4)

Get number of fields in result

```
int mssql_num_fields (int result)
```

Mssql_num_fields() returns the number of fields in a result set.

See also: **mssql_db_query()**, **mssql_query()**, **mssql_fetch_field()**, and **mssql_num_rows()**.

mssql_num_rows (PHP 3, PHP 4)

Get number of rows in result

```
int mssql_num_rows (string result)
```

Mssql_num_rows() returns the number of rows in a result set.

See also: **mssql_db_query()**, **mssql_query()**, and **mssql_fetch_row()**.

mssql_pconnect (PHP 3, PHP 4)

Open persistent MS SQL connection

```
int mssql_pconnect ([string servername [, string username [, string password] ]])
```

Returns: A positive MS SQL persistent link identifier on success, or false on error.

Mssql_pconnect() acts very much like **mssql_connect()** with two major differences.

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 (**mssql_close()** will not close links established by **mssql_pconnect()**).

This type of links is therefore called 'persistent'.

mssql_query (PHP 3, PHP 4)

Send MS SQL query

```
int mssql_query (string query [, int link_identifier])
```

Returns: A positive MS SQL result identifier on success, or false on error.

Mssql_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 **mssql_connect()** was called, and use it.

See also: **mssql_db_query()**, **mssql_select_db()**, and **mssql_connect()**.

mssql_result (PHP 3, PHP 4)

Get result data

```
int mssql_result (int result, int i, mixed field)
```

Mssql_result() returns the contents of one cell from a MS SQL result set. The field argument can be the field's offset, the field's name or the field's table dot field's name (tablename.fieldname). If the column name has been aliased ('select foo as bar from...'), it uses the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than **mssql_result()**. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: **mssql_fetch_row()**, **mssql_fetch_array()**, and **mssql_fetch_object()**.

mssql_select_db (PHP 3, PHP 4)

Select MS SQL database

```
int mssql_select_db (string database_name [, int link_identifier])
```

Returns: true on success, false on error

Mssql_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if **mssql_connect()** was called, and use it.

Every subsequent call to **mssql_query()** will be made on the active database.

See also: **mssql_connect()**, **mssql_pconnect()**, and **mssql_query()**

XLI. Miscellaneous functions

These functions were placed here because none of the other categories seemed to fit.

connection_aborted (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Returns true if client disconnected

```
int connection_aborted (void )
```

Returns true if client disconnected. See the [Connection Handling](#) description in the [Features](#) chapter for a complete explanation.

connection_status (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Returns connection status bitfield

```
int connection_status (void )
```

Returns the connection status bitfield. See the [Connection Handling](#) description in the [Features](#) chapter for a complete explanation.

connection_timeout (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Return true if script timed out

```
int connection_timeout (void )
```

Returns true if script timed out. See the [Connection Handling](#) description in the [Features](#) chapter for a complete explanation.

define (PHP 3, PHP 4)

Defines a named constant.

```
int define (string name, mixed value [, int case_insensitive])
```

Defines a named constant, which is similar to a variable except:

- Constants do not have a dollar sign '\$' before them;
- Constants may be accessed anywhere without regard to variable scoping rules;
- Constants may not be redefined or undefined once they have been set; and
- Constants may only evaluate to scalar values.

The name of the constant is given by *name*; the value is given by *value*.

The optional third parameter *case_insensitive* is also available. If the value *I* is given, then the constant will be defined case-insensitive. The default behaviour is case-sensitive; i.e. CONSTANT and Constant represent different values.

Example 1. Defining Constants

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

Define() returns TRUE on success and FALSE if an error occurs.

See also **defined()** and the section on [Constants](#).

defined (PHP 3, PHP 4)

Checks whether a given named constant exists

```
int defined (string name)
```

Returns true if the named constant given by *name* has been defined, false otherwise.

Example 1. Checking Constants

```
<?php
if (defined("CONSTANT")){ // Note that it should be quoted
    echo CONSTANT; //
}
?>
```

See also **define()** and the section on [Constants](#).

die (unknown)

Output a message and terminate the current script

```
void die (string message)
```

This language construct outputs a message and terminates parsing of the script. It does not return anything.

Example 1. die example

```
<?php
$filename = '/path/to/data-file';
$file = fopen ($filename, 'r')
    or die("unable to open file ($filename)");
?>
```

See also **exit()**.

eval (unknown)

Evaluate a string as PHP code

```
mixed eval (string code_str)
```

eval() evaluates the string given in *code_str* as PHP code. Among other things, this can be useful for storing code in a database text field for later execution.

There are some factors to keep in mind when using **eval()**. Remember that the string passed must be valid PHP code, including things like terminating statements with a semicolon so the parser doesn't die on the line after the **eval()**, and properly escaping things in *code_str*.

Also remember that variables given values under **eval()** will retain these values in the main script afterwards.

A return statement will terminate the evaluation of the string immediatley. In PHP 4 you may use `return` to return a value that will become the result of the **eval()** function while in PHP 3 **eval()** was of type `void` and did never return anything.

Example 1. Eval() example - simple text merge

```
<?php
$string = 'cup';
$name = 'coffee';
$str = 'This is a $string with my $name in it.<br>';
echo $str;
eval ("\$str = \"$str\";");
echo $str;
?>
```

The above example will show:

```
This is a $string with my $name in it.
This is a cup with my coffee in it.
```

exit (unknown)

Terminate current script

```
void exit(void);
```

This language construct terminates parsing of the script. It does not return.

See also **die()**.

get_browser (PHP 3, PHP 4)

Tells what the user's browser is capable of

```
object get_browser ([string user_agent])
```

`get_browser()` attempts to determine the capabilities of the user's browser. This is done by looking up the browser's information in the `browscap.ini` file. By default, the value of `$HTTP_USER_AGENT` is used; however, you can alter this (i.e., look up another browser's info) by passing the optional `user_agent` parameter to `get_browser()`.

The information is returned in an object, which will contain various data elements representing, for instance, the browser's major and minor version numbers and ID string; true/false values for features such as frames, JavaScript, and cookies; and so forth.

While `browscap.ini` contains information on many browsers, it relies on user updates to keep the database current. The format of the file is fairly self-explanatory.

The following example shows how one might list all available information retrieved about the user's browser.

Example 1. Get_browser() example

```
<?php
function list_array ($array) {
    while (list ($key, $value) = each ($array)) {
        $str .= "<b>$key:</b> $value<br>\n";
    }
    return $str;
}
echo "$HTTP_USER_AGENT<hr>\n";
$browser = get_browser();
echo list_array ((array) $browser);
?>
```

The output of the above script would look something like this:

```
Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)<hr>
<b>browser_name_pattern:</b> Mozilla/4\5.*<br>
<b>parent:</b> Netscape 4.0<br>
<b>platform:</b> Unknown<br>
<b>majorver:</b> 4<br>
<b>minorver:</b> 5<br>
<b>browser:</b> Netscape<br>
<b>version:</b> 4<br>
<b>frames:</b> 1<br>
<b>tables:</b> 1<br>
<b>cookies:</b> 1<br>
<b>backgroundsounds:</b> <br>
<b>vbscript:</b> <br>
<b>javascript:</b> 1<br>
<b>javaapplets:</b> 1<br>
<b>activexcontrols:</b> <br>
<b>beta:</b> <br>
<b>crawler:</b> <br>
<b>authenticodeupdate:</b> <br>
<b>msn:</b> <br>
```

In order for this to work, your `browscap` configuration file setting must point to the correct location of the `browscap.ini` file.

For more information (including locations from which you may obtain a `browscap.ini` file), check the PHP FAQ at <http://www.php.net/FAQ.php>.

highlight_file (PHP 4)

Syntax highlighting of a file

```
boolean highlight_file (string filename)
```

The **highlight_file()** function prints out a syntax highlighted version of the code contained in *filename* using the colors defined in the built-in syntax highlighter for PHP. It returns true on success, false otherwise (PHP 4).

Example 1. Creating a source highlighting URL

To setup a URL that can code hightlight any script that you pass to it, we will make use of the "ForceType" directive in Apache to generate a nice URL pattern, and use the function **highlight_file()** to show a nice looking code list.

In your httpd.conf you can add the following:

```
<Location /source>
    ForceType application/x-httdp-php
</Location>
```

And then make a file named "source" and put it in your web root directory.

```
<HTML>
<HEAD>
<TITLE>Source Display</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<?php
    $script = getenv ("PATH_TRANSLATED");
    if(!$script) {
        echo "<BR><B>ERROR: Script Name needed</B><BR>";
    } else {
        if (ereg("(\\.php|\\.inc)$", $script)) {
            echo "<H1>Source of: $PATH_INFO</H1>\n<HR>\n";
            highlight_file($script);
        } else {
            echo "<H1>ERROR: Only PHP or include script names are allowed</H1>";
        }
    }
    echo "<HR>Processed: ".date("Y/M/d H:i:s",time());
?>
</BODY>
</HTML>
```

Then you can use an URL like the one below to display a colorized version of a script located in "/path/to/script.php" in your web site.

<http://your.server.com/source/path/to/script.php>

See also **highlight_string()**, **show_source()**.

highlight_string (PHP 4)

Syntax highlighting of a string

```
void highlight_string (string str)
```

The **highlight_string()** function prints out a syntax highlighted version of *str* using the colors defined in the built-in syntax highlighter for PHP. It returns true on success, false otherwise (PHP 4).

See also **highlight_file()**, **show_source()**.

ignore_user_abort (PHP 3>= 3.0.7, PHP 4 >= 4.0b4)

Set whether a client disconnect should abort script execution

```
int ignore_user_abort ([int setting])
```

This function sets whether a client disconnect should cause a script to be aborted. It will return the previous setting and can be called without an argument to not change the current setting and only return the current setting. See the [Connection Handling](#) section in the [Features](#) chapter for a complete description of connection handling in PHP.

iptcparse (PHP 3>= 3.0.6, PHP 4)

Parse a binary IPTC http://www.iptc.org/ block into single tags.

```
array iptcparse (string iptcblock)
```

This function parses a binary IPTC block into its single tags. It returns an array using the tagmarker as an index and the value as the value. It returns false on error or if no IPTC data was found. See **GetImageSize()** for a sample.

leak (PHP 3, PHP 4)

Leak memory

```
void leak (int bytes)
```

Leak() leaks the specified amount of memory.

This is useful when debugging the memory manager, which automatically cleans up "leaked" memory when each request is completed.

pack (PHP 3, PHP 4)

Pack data into binary string.

```
string pack (string format [, mixed args ...])
```

Pack given arguments into binary string according to *format*. Returns binary string containing data.

The idea to this function was taken from Perl and all formatting codes work the same as there, however, there are some formatting codes that are missing such as Perl's "u" format code. The format string consists of format codes followed by an optional repeater argument. The repeater argument can be either an integer value or * for repeating to the end of the input data. For a, A, h, H the repeat count specifies how many characters of one data argument are taken, for @ it is the absolute position where to put the next data, for everything else the repeat count specifies how many data arguments are consumed and packed into the resulting binary string. Currently implemented are

- a NUL-padded string
- A SPACE-padded string
- h Hex string, low nibble first
- H Hex string, high nibble first
- c signed char
- C unsigned char
- s signed short (always 16 bit, machine byte order)
- S unsigned short (always 16 bit, machine byte order)
- n unsigned short (always 16 bit, big endian byte order)
- v unsigned short (always 16 bit, little endian byte order)
- i signed integer (machine dependent size and byte order)
- I unsigned integer (machine dependent size and byte order)
- l signed long (always 32 bit, machine byte order)
- L unsigned long (always 32 bit, machine byte order)
- N unsigned long (always 32 bit, big endian byte order)
- V unsigned long (always 32 bit, little endian byte order)
- f float (machine dependent size and representation)
- d double (machine dependent size and representation)
- x NUL byte
- X Back up one byte
- @ NUL-fill to absolute position

Example 1. Pack() format string

```
$binarydata = pack ("nvc*", 0x1234, 0x5678, 65, 66);
```

The resulting binary string will be 6 bytes long and contain the byte sequence 0x12, 0x34, 0x78, 0x56, 0x41, 0x42.

Note that the distinction between signed and unsigned values only affects the function **unpack()**, where as function **pack()** gives the same result for signed and unsigned format codes.

Also note that PHP internally stores integral values as signed values of a machine dependent size. If you give it an unsigned integral value too large to be stored that way it is converted to a double which often yields an undesired result.

show_source (PHP 4)

Syntax highlighting of a file

```
boolean show_source (string filename)
```

The **show_source()** function prints out a syntax highlighted version of the code contained in *filename* using the colors defined in the built-in syntax highlighter for PHP. It returns true on success, false otherwise (PHP 4).

Note: This function is an alias for the function **highlight_file()**

See also **highlight_string()**, **highlight_file()**.

sleep (PHP 3, PHP 4)

Delay execution

```
void sleep (int seconds)
```

The sleep function delays program execution for the given number of *seconds*.

See also **usleep()**.

uniqid (PHP 3, PHP 4)

Generate a unique id

```
int uniqid (string prefix [, boolean lcg])
```

Uniqid() returns a prefixed unique identifier based on the current time in microseconds. The prefix can be useful for instance if you generate identifiers simultaneously on several hosts that might happen to generate the identifier at the same microsecond. *Prefix* can be up to 114 characters long.

If the optional *lcg* parameter is true, **uniqid()** will add additional "combined LCG" entropy at the end of the return value, which should make the results more unique.

With an empty *prefix*, the returned string will be 13 characters long. If *lcg* is true, it will be 23 characters.

Note: The *lcg* parameter is only available in PHP 4 and PHP 3.0.13 and later.

If you need a unique identifier or token and you intend to give out that token to the user via the network (i.e. session cookies), it is recommended that you use something along the lines of

```
$token = md5(uniqid("")); // no random portion
$better_token = md5(uniqid(rand())); // better, difficult to guess
```

This will create a 32 character identifier (a 128 bit hex number) that is extremely difficult to predict.

unpack (PHP 3, PHP 4)

Unpack data from binary string

```
array unpack (string format, string data)
```

Unpack() from binary string into array according to *format*. Returns array containing unpacked elements of binary string.

Unpack() works slightly different from Perl as the unpacked data is stored in an associative array. To accomplish this you have to name the different format codes and separate them by a slash /.

Example 1. Unpack() format string

```
$array = unpack ("c2chars/nint", $binarydata);
```

The resulting array will contain the entries "chars1", "chars2" and "int".

For an explanation of the format codes see also: **pack()**

Note that PHP internally stores integral values as signed. If you unpack a large unsigned long and it is of the same size as PHP internally stored values the result will be a negative number even though unsigned unpacking was specified.

usleep (PHP 3, PHP 4)

Delay execution in microseconds

```
void usleep (int micro_seconds)
```

The **usleep()** function delays program execution for the given number of *micro_seconds*.

See also **sleep()**.

Note: This function does not work on Windows systems.

XLII. mSQL functions

These functions allow you to access mSQL database servers. In order to have these functions available, you must compile php with msql support by using the `-with-msql[=dir]` option. The default location is `/usr/local/Hughes`.

More information about mSQL can be found at <http://www.hughes.com.au/>.

msql (PHP 3, PHP 4)

Send mSQL query

```
int msql (string database, string query, int link_identifier)
```

Returns a positive mSQL query identifier to the query result, or false on error.

msql() selects a database and executes a query on it. If the optional link identifier isn't specified, the function will try to find an open link to the mSQL server and if no such link is found it'll try to create one as if **msql_connect()** was called with no arguments (see **msql_connect()**).

msql_affected_rows (PHP 3>= 3.0.6, PHP 4)

Returns number of affected rows

```
int msql_affected_rows (int query_identifier)
```

Returns number of affected ("touched") rows by a specific query (i.e. the number of rows returned by a SELECT, the number of rows modified by an update, or the number of rows removed by a delete).

See also: **msql_query()**.

msql_close (PHP 3, PHP 4)

Close mSQL connection

```
int msql_close (int link_identifier)
```

Returns true on success, false on error.

msql_close() closes the link to a mSQL 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.

msql_close() will not close persistent links generated by **msql_pconnect()**.

See also: **msql_connect()** and **msql_pconnect()**.

msql_connect (PHP 3, PHP 4)

Open mSQL connection

```
int msql_connect ([string hostname [, string hostname[:port] [, string username [, string password]]]])
```

Returns a positive mSQL link identifier on success, or false on error.

msql_connect() establishes a connection to a mSQL server. The hostname argument is optional, and if it's missing, localhost is assumed.

In case a second call is made to **msql_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 **msql_close()**.

See also **msql_pconnect()**, **msql_close()**.

msql_create_db (PHP 3, PHP 4)

Create mSQL database

```
int msql_create_db (string database name [, int link_identifier])
```

msql_create_db() attempts to create a new database on the server associated with the specified link identifier.

See also: **msql_drop_db()**.

msql_createdb (PHP 3, PHP 4)

Create mSQL database

```
int msql_createdb (string database name [, int link_identifier])
```

Identical to **msql_create_db()**.

msql_data_seek (PHP 3, PHP 4)

Move internal row pointer

```
int msql_data_seek (int query_identifier, int row_number)
```

Returns true on success, false on failure.

msql_data_seek() moves the internal row pointer of the mSQL result associated with the specified query identifier to pointer to the specified row number. The next call to **msql_fetch_row()** would return that row.

See also: **msql_fetch_row()**.

msql_dbname (PHP 3, PHP 4)

Get current mSQL database name

```
string msql_dbname (int query_identifier, int i)
```

msql_dbname() returns the database name stored in position *i* of the result pointer returned from the **msql_listdbs()** function. The **msql_numrows()** function can be used to determine how many database names are available.

mSQL_drop_db (PHP 3, PHP 4)

Drop (delete) mSQL database

```
int msql_drop_db (string database_name, int link_identifier)
```

Returns true on success, false on failure.

mSQL_drop_db() attempts to drop (remove) an entire database from the server associated with the specified link identifier.

See also: **mSQL_create_db()**.

mSQL_dropdb (PHP 3, PHP 4)

Drop (delete) mSQL database

See **mSQL_drop_db()**.

mSQL_error (PHP 3, PHP 4)

Returns error message of last mSQL call

```
string msql_error ()
```

Errors coming back from the mSQL database backend no longer issue warnings. Instead, use these functions to retrieve the error string.

mSQL_fetch_array (PHP 3, PHP 4)

Fetch row as array

```
int msql_fetch_array (int query_identifier [, int result_type])
```

Returns an array that corresponds to the fetched row, or false if there are no more rows.

mSQL_fetch_array() is an extended version of **mSQL_fetch_row()**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

The second optional argument *result_type* in **mSQL_fetch_array()** is a constant and can take the following values: MSQL_ASSOC, MSQL_NUM, and MYSQL_BOTH.

Be careful if you are retrieving results from a query that may return a record that contains only one field that has a value of 0 (or an empty string, or NULL).

An important thing to note is that using **mSQL_fetch_array()** is NOT significantly slower than using **mSQL_fetch_row()**, while it provides a significant added value.

For further details, also see **mSQL_fetch_row()**.

mSQL_fetch_field (PHP 3, PHP 4)

Get field information

```
object msql_fetch_field (int query_identifier, int field_offset)
```

Returns an object containing field information

msql_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retrieved by **msql_fetch_field()** is retrieved.

The properties of the object are:

- name - column name
- table - name of the table the column belongs to
- not_null - 1 if the column cannot be null
- primary_key - 1 if the column is a primary key
- unique - 1 if the column is a unique key
- type - the type of the column

See also **msql_field_seek()**.

msql_fetch_object (PHP 3, PHP 4)

Fetch row as object

```
int msql_fetch_object (int query_identifier [, int result_type])
```

Returns an object with properties that correspond to the fetched row, or false if there are no more rows.

msql_fetch_object() is similar to **msql_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The optional second argument *result_type* in **msql_fetch_array()** is a constant and can take the following values: **MSQL_ASSOC**, **MSQL_NUM**, and **MSQL_BOTH**.

Speed-wise, the function is identical to **msql_fetch_array()**, and almost as quick as **msql_fetch_row()** (the difference is insignificant).

See also: **msql_fetch_array()** and **msql_fetch_row()**.

msql_fetch_row (PHP 3, PHP 4)

Get row as enumerated array

```
array msql_fetch_row (int query_identifier)
```

Returns an array that corresponds to the fetched row, or false if there are no more rows.

msql_fetch_row() fetches one row of data from the result associated with the specified query identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to **msql_fetch_row()** would return the next row in the result set, or false if there are no more rows.

See also: **msql_fetch_array()**, **msql_fetch_object()**, **msql_data_seek()**, and **msql_result()**.

msql_fieldname (PHP 3, PHP 4)

Get field name

```
string msql_fieldname (int query_identifier, int field)
```

msql_fieldname() returns the name of the specified field. *query_identifier* is the query identifier, and *field* is the field index. `mysql_fieldname($result, 2);` will return the name of the second field in the result associated with the result identifier.

msql_field_seek (PHP 3, PHP 4)

Set field offset

```
int msql_field_seek (int query_identifier, int field_offset)
```

Seeks to the specified field offset. If the next call to **mysql_fetch_field()** won't include a field offset, this field would be returned.

See also: **mysql_fetch_field()**.

msql_fieldtable (PHP 3, PHP 4)

Get table name for field

```
int msql_fieldtable (int query_identifier, int field)
```

Returns the name of the table *field* was fetched from.

msql_fieldtype (PHP 3, PHP 4)

Get field type

```
string msql_fieldtype (int query_identifier, int i)
```

msql_fieldtype() is similar to the **mysql_fieldname()** function. The arguments are identical, but the field type is returned. This will be one of "int", "char" or "real".

msql_fieldflags (PHP 3, PHP 4)

Get field flags

```
string msql_fieldflags (int query_identifier, int i)
```

msql_fieldflags() returns the field flags of the specified field. Currently this is either, "not null", "primary key", a combination of the two or "" (an empty string).

mysql_fieldlen (PHP 3, PHP 4)

Get field length

```
int mysql_fieldlen (int query_identifier, int i)
```

mysql_fieldlen() returns the length of the specified field.

mysql_free_result (PHP 3, PHP 4)

Free result memory

```
int mysql_free_result (int query_identifier)
```

mysql_free_result() frees the memory associated with *query_identifier*. When PHP completes a request, this memory is freed automatically, so you only need to call this function when you want to make sure you don't use too much memory while the script is running.

mysql_free_result (PHP 3, PHP 4)

Free result memory

See **mysql_free_result()**

mysql_list_fields (PHP 3, PHP 4)

List result fields

```
int mysql_list_fields (string database, string tablename)
```

mysql_list_fields() retrieves information about the given tablename. Arguments are the database name and the table name. A result pointer is returned which can be used with **mysql_fieldflags()**, **mysql_fieldlen()**, **mysql_fieldname()**, and **mysql_fieldtype()**. A query identifier is a positive integer. The function returns -1 if an error occurs. A string describing the error will be placed in \$phperrormsg, and unless the function was called as @mysql_list_fields() then this error string will also be printed out.

See also **mysql_error()**.

mysql_listfields (PHP 3, PHP 4)

List result fields

See **mysql_list_fields()**.

mysql_list_dbs (PHP 3, PHP 4)

List mSQL databases on server

```
int msql_list_dbs(void);
```

msql_list_dbs() will return a result pointer containing the databases available from the current msql daemon. Use the **msql_dbname()** function to traverse this result pointer.

msql_listdbs (PHP 3, PHP 4)

List mSQL databases on server

See **msql_list_dbs()**.

msql_list_tables (PHP 3, PHP 4)

List tables in an mSQL database

```
int msql_list_tables (string database)
```

msql_list_tables() takes a database name and result pointer much like the **mysql()** function. The **msql_tablename()** function should be used to extract the actual table names from the result pointer.

msql_listtables (PHP 3, PHP 4)

List tables in an mSQL database

See **msql_list_tables()**.

msql_num_fields (PHP 3, PHP 4)

Get number of fields in result

```
int msql_num_fields (int query_identifier)
```

msql_num_fields() returns the number of fields in a result set.

See also: **mysql()**, **mysql_query()**, **mysql_fetch_field()**, and **mysql_num_rows()**.

msql_num_rows (PHP 3, PHP 4)

Get number of rows in result

```
int msql_num_rows (int query_identifier)
```

MsqNumRows() returns the number of rows in a result set.

See also: **mysql()**, **mysql_query()**, and **mysql_fetch_row()**.

msql_numfields (PHP 3, PHP 4)

Get number of fields in result

```
int msql_numfields (int query_identifier)
```

Identical to **mysql_num_fields()**.

msql_numrows (PHP 3, PHP 4)

Get number of rows in result

```
int msql_numrows(void);
```

Identical to **mysql_num_rows()**.

msql_pconnect (PHP 3, PHP 4)

Open persistent mSQL connection

```
int msql_pconnect ([string hostname [, string hostname[:port] [, string username [, string password]]]])
```

Returns a positive mSQL persistent link identifier on success, or false on error.

mysql_pconnect() acts very much like **mysql_connect()** with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host. 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 (**mysql_close()** will not close links established by **mysql_pconnect()**).

This type of links is therefore called 'persistent'.

msql_query (PHP 3, PHP 4)

Send mSQL query

```
int msql_query (string query, int link_identifier)
```

mysql_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 **mysql_connect()** was called, and use it.

Returns a positive mSQL query identifier on success, or false on error.

See also: **mysql()**, **mysql_select_db()**, and **mysql_connect()**.

msql_regcase (PHP 3, PHP 4)

Make regular expression for case insensitive match

See [sql_regcase\(\)](#).

msql_result (PHP 3, PHP 4)

Get result data

```
int msql_result (int query_identifier, int i, mixed field)
```

Returns the contents of the cell at the row and offset in the specified mSQL result set.

msql_result() returns the contents of one cell from a mSQL result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from ...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than **msql_result()**. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: **msql_fetch_row()**, **msql_fetch_array()**, and **msql_fetch_object()**.

msql_select_db (PHP 3, PHP 4)

Select mSQL database

```
int msql_select_db (string database_name, int link_identifier)
```

Returns true on success, false on error.

msql_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if **msql_connect()** was called, and use it.

Every subsequent call to **msql_query()** will be made on the active database.

See also: **msql_connect()**, **msql_pconnect()**, and **msql_query()**.

msql_selectdb (PHP 3, PHP 4)

Select mSQL database

See **msql_select_db()**.

msql_tablename (PHP 3, PHP 4)

Get table name of field

```
string msql_tablename (int query_identifier, int field)
```

mssql_tablename() takes a result pointer returned by the **mssql_list_tables()** function as well as an integer index and returns the name of a table. The **mssql_numrows()** function may be used to determine the number of tables in the result pointer.

Example 1. mssql_tablename() example

```
<?php
mssql_connect ("localhost");
$result = mssql_list_tables ("wisconsin");
$i = 0;
while ($i < mssql_numrows ($result)) {
    $tb_names[$i] = mssql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

XLIII. MySQL functions

These functions allow you to access MySQL database servers. In order to have these functions available, you must compile php with mysql support by using the `-with-mysql` option. If you use this option without specifying the path to mysql, php will use the built-in mysql client libraries. Users who run other applications that use mysql (for example, running php3 and php4 as concurrent apache modules, or auth-mysql) should always specify the path to mysql:

`-with-mysql=/path/to/mysql`. This will force php to use the client libraries installed by mysql, avoiding any conflicts.

More information about MySQL can be found at <http://www.mysql.com/>.

Documentation for MySQL can be found at <http://www.mysql.com/documentation/>.

mysql_affected_rows (PHP 3, PHP 4)

Get number of affected rows in previous MySQL operation

```
int mysql_affected_rows ([int link_identifier])
```

mysql_affected_rows() returns the number of rows affected by the last INSERT, UPDATE or DELETE query on the server associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note: If you are using transactions, you need to call **mysql_affected_rows()** after your INSERT, UPDATE, or DELETE query, not after the commit.

If the last query was a DELETE query with no WHERE clause, all of the records will have been deleted from the table but this function will return zero.

This command is not effective for SELECT statements, only on statements which modify records. To retrieve the number of rows returned from a SELECT, use **mysql_num_rows()**.

If the last query failed, this function will return -1.

mysql_change_user (PHP 3>= 3.0.13)

Change logged in user of the active connection

```
int mysql_change_user (string user, string password [, string database [, int link_identifier]])
```

mysql_change_user() changes the logged in user of the current active connection, or the connection given by the optional parameter *link_identifier*. If a database is specified, this will default or current database after the user has been changed. If the new user and password authorization fails, the current connected user stays active.

Note: This function was introduced in PHP 3.0.13 and requires MySQL 3.23.3 or higher.

mysql_close (PHP 3, PHP 4)

Close MySQL connection

```
int mysql_close ([int link_identifier])
```

Returns: true on success, false on error.

mysql_close() closes the connection to the MySQL server that's associated with the specified link identifier. If *link_identifier* isn't specified, the last opened link is used.

Using **mysql_close()** isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

Note: **mysql_close()** will not close persistent links created by **mysql_pconnect()**.

Example 1. MySQL close example

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret")
or die ("Could not connect");
print ("Connected successfully");
mysql_close ($link);
?>
```

See also: **[mysql_connect\(\)](#)**, and **[mysql_pconnect\(\)](#)**.

mysql_connect (PHP 3, PHP 4)

Open a connection to a MySQL Server

```
int mysql_connect ([string hostname [:port] [:/path/to/socket] [, string username [, string password]]])
```

Returns a positive MySQL link identifier on success, or an error message on failure.

mysql_connect() establishes a connection to a MySQL server. The following defaults are assumed for missing optional parameters: *host : port* = 'localhost:3306', *username* = name of the user that owns the server process and *password* = empty password.

The hostname string can also include a port number. eg. "hostname:port" or a path to a socket eg. ":/path/to/socket" for the localhost.

Note: Support for ":port" was added in PHP 3.0B4.

Support for ":/path/to/socket" was added in PHP 3.0.10.

You can suppress the error message on failure by prepending '@' to the function name.

If a second call is made to **mysql_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 **mysql_close()**.

Example 1. MySQL connect example

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret")
or die ("Could not connect");
print ("Connected successfully");
mysql_close ($link);
?>
```

See also **[mysql_pconnect\(\)](#)**, and **[mysql_close\(\)](#)**.

mysql_create_db (PHP 3, PHP 4)

Create a MySQL database

```
int mysql_create_db (string database name [, int link_identifier])
```

mysql_create_db() attempts to create a new database on the server associated with the specified link identifier.

Example 1. MySQL create database example

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim")
or die ("Could not connect");
if (mysql_create_db ("my_db")) {
    print ("Database created successfully\n");
} else {
    printf ("Error creating database: %s\n", mysql_error ());
}
?>
```

For downwards compatibility **mysql_createdb()** can also be used.

See also: **mysql_drop_db()**.

mysql_data_seek (PHP 3, PHP 4)

Move internal result pointer

```
int mysql_data_seek (int result_identifier, int row_number)
```

Returns: true on success, false on failure.

mysql_data_seek() moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to **mysql_fetch_row()** would return that row.

Row_number starts at 0.

Example 1. MySQL data seek example

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim")
or die ("Could not connect");

mysql_select_db ("samp_db")
or die ("Could not select database");

$query = "SELECT last_name, first_name FROM friends";
$result = mysql_query ($query)
or die ("Query failed");

# fetch rows in reverse order

for ($i = mysql_num_rows ($result) - 1; $i >= 0; $i--) {
    if (!mysql_data_seek ($result, $i)) {
        printf ("Cannot seek to row %d\n", $i);
        continue;
    }

    if (!$row = mysql_fetch_object ($result)))
        continue;

    printf ("%s %s<BR>\n", $row->last_name, $row->first_name);
```

```

    }

    mysql_free_result ($result);
?>

```

mysql_db_name (PHP 3>= 3.0.6, PHP 4)

Get result data

```
int mysql_db_name (int result, int row [, mixed field])
```

mysql_db_name() takes as its first parameter the result pointer from a call to **mysql_list_dbs()**. The *row* parameter is an index into the result set.

If an error occurs, FALSE is returned. Use **mysql_errno()** and **mysql_error()** to determine the nature of the error.

Example 1. Mysql_db_name() example

```
<?php
error_reporting(E_ALL);

mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs();

$i = 0;
$cnt = mysql_num_rows($db_list);
while ($i < $cnt) {
    echo mysql_db_name($db_list, $i) . "\n";
    $i++;
}
?>
```

For backward compatibility, **mysql_dbname()** is also accepted. This is deprecated, however.

mysql_db_query (PHP 3, PHP 4)

Send a MySQL query

```
int mysql_db_query (string database, string query [, int link_identifier])
```

Returns: A positive MySQL result identifier to the query result, or false on error.

mysql_db_query() selects a database and executes a query on it. If the optional link identifier isn't specified, the function will try to find an open link to the MySQL server and if no such link is found it'll try to create one as if **mysql_connect()** was called with no arguments

See also **mysql_connect()**.

For downwards compatibility **mysql()** can also be used.

mysql_drop_db (PHP 3, PHP 4)

Drop (delete) a MySQL database

```
int mysql_drop_db (string database_name [, int link_identifier])
```

Returns: true on success, false on failure.

mysql_drop_db() attempts to drop (remove) an entire database from the server associated with the specified link identifier.

See also: **mysql_create_db()**. For downward compatibility **mysql_dropdb()** can also be used.

mysql_errno (PHP 3, PHP 4)

Returns the numerical value of the error message from previous MySQL operation

```
int mysql_errno ([int link_identifier])
```

Returns the error number from the last mySQL function, or 0 (zero) if no error occurred.

Errors coming back from the mySQL database backend no longer issue warnings. Instead, use **mysql_errno()** to retrieve the error code. Note that this function only returns the error code from the most recently executed mySQL function (not including **mysql_error()** and **mysql_errno()**), so if you want to use it, make sure you check the value before calling another mySQL function.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().".".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().".".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().".".mysql_error()."<BR>";
?>
```

See also: **mysql_error()**

mysql_error (PHP 3, PHP 4)

Returns the text of the error message from previous MySQL operation

```
string mysql_error ([int link_identifier])
```

Returns the error text from the last mySQL function, or "" (the empty string) if no error occurred.

Errors coming back from the mySQL database backend no longer issue warnings. Instead, use **mysql_error()** to retrieve the error text. Note that this function only returns the error text from the most recently executed mySQL function (not including **mysql_error()** and **mysql_errno()**), so if you want to use it, make sure you check the value before calling another mySQL function.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().".".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().".".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().".".mysql_error()."<BR>";
?>
```

See also: **mysql_errno()**

mysql_fetch_array (PHP 3, PHP 4)

Fetch a result row as an associative array, a numeric array, or both.

```
array mysql_fetch_array (int result [, int result_type])
```

Returns an array that corresponds to the fetched row, or false if there are no more rows.

mysql_fetch_array() is an extended version of **mysql_fetch_row()**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use the numeric index of the column or make an alias for the column.

```
select t1.f1 as foo t2.f1 as bar from t1, t2
```

An important thing to note is that using **mysql_fetch_array()** is NOT significantly slower than using **mysql_fetch_row()**, while it provides a significant added value.

The optional second argument *result_type* in **mysql_fetch_array()** is a constant and can take the following values: MYSQL_ASSOC, MYSQL_NUM, and MYSQL_BOTH. (This feature was added in PHP 3.0.7)

For further details, see also **mysql_fetch_row()** and **mysql_fetch_assoc()**.

Example 1. Mysql_fetch_array()

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database", "select user_id, fullname from table");
while ($row = mysql_fetch_array ($result)) {
    echo "user_id: ".$row["user_id"]."<br>\n";
    echo "user_id: ".$row[0]."<br>\n";
    echo "fullname: ".$row["fullname"]."<br>\n";
    echo "fullname: ".$row[1]."<br>\n";
}
mysql_free_result ($result);
?>
```

mysql_fetch_assoc (PHP 4 >= 4.0.3)

Fetch a result row as an associative array

```
array mysql_fetch_assoc (int result)
```

Returns an associative array that corresponds to the fetched row, or false if there are no more rows.

mysql_fetch_assoc() is equivalent to calling **mysql_fetch_array()** with MYSQL_ASSOC for the optional second parameter. It only returns an associative array. This is the way **mysql_fetch_array()** originally worked. If you need the numeric indices as well as the associative, use **mysql_fetch_array()**.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use **mysql_fetch_array()** and have it return the numeric indices as well.

An important thing to note is that using **mysql_fetch_assoc()** is NOT significantly slower than using **mysql_fetch_row()**, while it provides a significant added value.

For further details, see also **mysql_fetch_row()** and **mysql_fetch_array()**.

Example 1. Mysql_fetch_assoc()

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database","select * from table");
while ($row = mysql_fetch_assoc ($result)) {
    echo $row["user_id"];
    echo $row["fullname"];
}
mysql_free_result ($result);
?>
```

mysql_fetch_field (PHP 3, PHP 4)

Get column information from a result and return as an object

```
object mysql_fetch_field (int result [, int field_offset])
```

Returns an object containing field information.

mysql_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retrieved by **mysql_fetch_field()** is retrieved.

The properties of the object are:

- name - column name
- table - name of the table the column belongs to
- max_length - maximum length of the column
- not_null - 1 if the column cannot be null
- primary_key - 1 if the column is a primary key
- unique_key - 1 if the column is a unique key
- multiple_key - 1 if the column is a non-unique key
- numeric - 1 if the column is numeric
- blob - 1 if the column is a BLOB
- type - the type of the column
- unsigned - 1 if the column is unsigned
- zerofill - 1 if the column is zero-filled

Example 1. Mysql_fetch_field()

```
<?php
mysql_connect ($host, $user, $password)
    or die ("Could not connect");
$result = mysql_db_query ("database", "select * from table")
    or die ("Query failed");
# get column metadata
$i = 0;
while ($i < mysql_num_fields ($result)) {
    echo "Information for column $i:<BR>\n";
    $meta = mysql_fetch_field ($result);
    if (!$meta) {
        echo "No information available<BR>\n";
    }
    echo "<PRE>
blob:           $meta->blob
max_length:     $meta->max_length
multiple_key:   $meta->multiple_key
name:           $meta->name
not_null:       $meta->not_null
numeric:        $meta->numeric
primary_key:   $meta->primary_key
table:          $meta->table
type:           $meta->type
unique_key:    $meta->unique_key
unsigned:       $meta->unsigned
zerofill:       $meta->zerofill
</PRE>";
    $i++;
}
mysql_free_result ($result);
?>
```

See also **mysql_field_seek()**.

mysql_fetch_lengths (PHP 3, PHP 4)

Get the length of each output in a result

```
array mysql_fetch_lengths ( int result )
```

Returns: An array that corresponds to the lengths of each field in the last row fetched by **mysql_fetch_row()**, or false on error.

mysql_fetch_lengths() stores the lengths of each result column in the last row returned by **mysql_fetch_row()**, **mysql_fetch_array()**, and **mysql_fetch_object()** in an array, starting at offset 0.

See also: **mysql_fetch_row()**.

mysql_fetch_object (PHP 3, PHP 4)

Fetch a result row as an object

```
object mysql_fetch_object ( int result [, int result_type] )
```

Returns an object with properties that correspond to the fetched row, or false if there are no more rows.

mysql_fetch_object() is similar to **mysql_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The optional argument *result_type* is a constant and can take the following values: MYSQL_ASSOC, MYSQL_NUM, and MYSQL_BOTH.

Speed-wise, the function is identical to **mysql_fetch_array()**, and almost as quick as **mysql_fetch_row()** (the difference is insignificant).

Example 1. mysql_fetch_object() example

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database", "select * from table");
while ($row = mysql_fetch_object ($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result ($result);
?>
```

See also: **mysql_fetch_array()** and **mysql_fetch_row()**.

mysql_fetch_row (PHP 3, PHP 4)

Get a result row as an enumerated array

```
array mysql_fetch_row (int result)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

mysql_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 **mysql_fetch_row()** would return the next row in the result set, or false if there are no more rows.

See also: **mysql_fetch_array()**, **mysql_fetch_object()**, **mysql_data_seek()**, **mysql_fetch_lengths()**, and **mysql_result()**.

mysql_field_flags (PHP 3, PHP 4)

Get the flags associated with the specified field in a result

```
string mysql_field_flags (int result, int field_offset)
```

mysql_field_flags() returns the field flags of the specified field. The flags are reported as a single word per flag separated by a single space, so that you can split the returned value using **explode()**.

The following flags are reported, if your version of MySQL is current enough to support them: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".

For downward compatibility **mysql_fieldflags()** can also be used.

mysql_field_name (PHP 3, PHP 4)

Get the name of the specified field in a result

```
string mysql_field_name (int result, int field_index)
```

mysql_field_name() returns the name of the specified field index. *result* must be a valid result identifier and *field_index* is the numerical offset of the field.

Note: *field_index* starts at 0.

e.g. The index of the third field would actually be 2, the index of the fourth field would be 3 and so on.

Example 1. mysql_field_name() example

```
// The users table consists of three fields:  
//   user_id  
//   username  
//   password.  
  
$res = mysql_db_query("users", "select * from users", $link);  
  
echo mysql_field_name($res, 0) . "\n";  
echo mysql_field_name($res, 2);
```

The above example would produce the following output:

```
user_id  
password
```

For downwards compatibility **mysql_fieldname()** can also be used.

mysql_field_len (PHP 3, PHP 4)

Returns the length of the specified field

```
int mysql_field_len (int result, int field_offset)
```

mysql_field_len() returns the length of the specified field.

For downward compatibility **mysql_fieldlen()** can also be used.

mysql_field_seek (PHP 3, PHP 4)

Set result pointer to a specified field offset

```
int mysql_field_seek (int result, int field_offset)
```

Seeks to the specified field offset. If the next call to **mysql_fetch_field()** doesn't include a field offset, the field offset specified in **mysql_field_seek()** will be returned.

See also: **mysql_fetch_field()**.

mysql_field_table (PHP 3, PHP 4)

Get name of the table the specified field is in

```
string mysql_field_table (int result, int field_offset)
```

Returns the name of the table that the specified field is in.

For downward compatibility **mysql_fieldtable()** can also be used.

mysql_field_type (PHP 3, PHP 4)

Get the type of the specified field in a result

```
string mysql_field_type (int result, int field_offset)
```

mysql_field_type() is similar to the **mysql_field_name()** function. The arguments are identical, but the field type is returned instead. The field type will be one of "int", "real", "string", "blob", and others as detailed in the MySQL documentation (<http://www.mysql.com/documentation/>).

Example 1. mysql field types

```
<?php

mysql_connect ("localhost:3306");
mysql_select_db ("wisconsin");
$result = mysql_query ("SELECT * FROM onek");
$fields = mysql_num_fields ($result);
$rows   = mysql_num_rows ($result);
$i = 0;
$table = mysql_field_table ($result, $i);
echo "Your '$table' table has ".$fields." fields and ".$rows." records <BR>";
echo "The table has the following fields <BR>";
while ($i < $fields) {
    $type   = mysql_field_type  ($result, $i);
    $name   = mysql_field_name   ($result, $i);
    $len    = mysql_field_len    ($result, $i);
    $flags  = mysql_field_flags  ($result, $i);
    echo $type." ".$name." ".$len." ".$flags."<BR>";
    $i++;
}
mysql_close();

?>
```

For downward compatibility **mysql_fieldtype()** can also be used.

mysql_free_result (PHP 3, PHP 4)

Free result memory

```
int mysql_free_result (int result)
```

mysql_free_result() will free all memory associated with the result identifier *result*.

mysql_free_result() only needs to be called if you are concerned about how much memory is being used for queries that return large result sets. All associated result memory is automatically freed at the end of the script's execution.

For downward compatibility **mysql.freeresult()** can also be used.

mysql_insert_id (PHP 3, PHP 4)

Get the id generated from the previous INSERT operation

```
int mysql_insert_id ([int link_identifier])
```

mysql_insert_id() returns the ID generated for an AUTO_INCREMENT column by the previous INSERT query using the given *link_identifier*. If *link_identifier* isn't specified, the last opened link is assumed.

mysql_insert_id() returns 0 if the previous query does not generate an AUTO_INCREMENT value. If you need to save the value for later, be sure to call **mysql_insert_id()** immediately after the query that generates the value.

Note: The value of the MySQL SQL function `LAST_INSERT_ID()` always contains the most recently generated AUTO_INCREMENT value, and is not reset between queries.

Warning

mysql_insert_id() converts the return type of the native MySQL C API function `mysql_insert_id()` to a type of `long`. If your AUTO_INCREMENT column has a column type of BIGINT, the value returned by **mysql_insert_id()** will be incorrect. Instead, use the internal MySQL SQL function `LAST_INSERT_ID()`.

mysql_list_dbs (PHP 3, PHP 4)

List databases available on a MySQL server

```
int mysql_list_dbs ([int link_identifier])
```

mysql_list_dbs() will return a result pointer containing the databases available from the current mysql daemon. Use the **mysql_tablename()** function to traverse this result pointer.

Example 1. mysql_list_dbs() example

```
$link = mysql_connect('localhost', 'myname', 'secret');
$db_list = mysql_list_dbs($link);

while ($row = mysql_fetch_object($db_list)) {
    echo $row->Database . "\n";
}
```

The above example would produce the following output:

```
database1
database2
database3
...
```

Note: The above code would just as easily work with **mysql_fetch_row()** or other similar functions.

For downward compatibility **mysql_listdbs()** can also be used.

mysql_list_fields (PHP 3, PHP 4)

List MySQL result fields

```
int mysql_list_fields (string database_name, string table_name [, int link_identifier])
```

mysql_list_fields() retrieves information about the given tablename. Arguments are the database name and the table name. A result pointer is returned which can be used with **mysql_field_flags()**, **mysql_field_len()**, **mysql_field_name()**, and **mysql_field_type()**.

A result identifier is a positive integer. The function returns -1 if an error occurs. A string describing the error will be placed in \$phperrormsg, and unless the function was called as @mysql() then this error string will also be printed out.

Example 1. mysql_list_fields() example

```
$link = mysql_connect('localhost', 'myname', 'secret');

$fields = mysql_list_fields("database1", "table1", $link);
$columns = mysql_num_fields($fields);

for ($i = 0; $i < $columns; $i++) {
    echo mysql_field_name($fields, $i) . "\n";
}
```

The above example would produce the following output:

```
field1
field2
field3
...
```

For downward compatibility **mysql_listfields()** can also be used.

mysql_list_tables (PHP 3, PHP 4)

List tables in a MySQL database

```
int mysql_list_tables (string database [, int link_identifier])
```

mysql_list_tables() takes a database name and returns a result pointer much like the **mysql_db_query()** function. The **mysql_tablename()** function should be used to extract the actual table names from the result pointer.

For downward compatibility **mysql_listtables()** can also be used.

mysql_num_fields (PHP 3, PHP 4)

Get number of fields in result

```
int mysql_num_fields (int result)
```

mysql_num_fields() returns the number of fields in a result set.

See also: **mysql_db_query()**, **mysql_query()**, **mysql_fetch_field()**, **mysql_num_rows()**.

For downward compatibility **mysql_numfields()** can also be used.

mysql_num_rows (PHP 3, PHP 4)

Get number of rows in result

```
int mysql_num_rows (int result)
```

mysql_num_rows() returns the number of rows in a result set. This command is only valid for SELECT statements. To retrieve the number of rows returned from a INSERT, UPDATE or DELETE, use **mysql_affected_rows()**.

Example 1. mysql_num_rows example by crubel@trilizio.org ()

```
<?php
$conn = mysql_connect("hostaddress", "username", "password");
mysql_select_db("database",$conn); // needed if you have multiple db's
$Resultfornummembers = mysql_query("SELECT * FROM Accounts",$conn);
$NumMembers = mysql_num_rows($Resultfornummembers);
echo "$NumMembers Members";
?>
```

See also: **mysql_db_query()**, **mysql_query()** and, **mysql_fetch_row()**.

For downward compatibility **mysql_numrows()** can also be used.

mysql_pconnect (PHP 3, PHP 4)

Open a persistent connection to a MySQL Server

```
int mysql_pconnect ([string hostname [:port] [:/path/to/socket] [, string username [, string password]]])
```

Returns: A positive MySQL persistent link identifier on success, or false on error.

mysql_pconnect() establishes a connection to a MySQL server. The following defaults are assumed for missing optional parameters: *host :port* = 'localhost:3306', *username* = name of the user that owns the server process and *password* = empty password.

The hostname string can also include a port number. eg. "hostname:port" or a path to a socket eg. ":/path/to/socket" for the localhost.

Note: Support for ":port" was added in 3.0B4.

Support for the ":/path/to/socket" was added in 3.0.10.

mysql_pconnect() acts very much like **mysql_connect()** with two major differences.

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 (**mysql_close()** will not close links established by **mysql_pconnect()**).

This type of links is therefore called 'persistent'.

mysql_query (PHP 3, PHP 4)

Send a MySQL query

```
int mysql_query (string query [, int link_identifier])
```

mysql_query() sends a query to the currently active database on the server that's associated with the specified link identifier. If *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 **mysql_connect()** was called with no arguments, and use it.

Note: The query string should not end with a semicolon.

mysql_query() returns TRUE (non-zero) or FALSE to indicate whether or not the query succeeded. A return value of TRUE means that the query was legal and could be executed by the server. It does not indicate anything about the number of rows affected or returned. It is perfectly possible for a query to succeed but affect no rows or return no rows.

The following query is syntactically invalid, so **mysql_query()** fails and returns FALSE:

Example 1. mysql_query()

```
<?php
$result = mysql_query ("SELECT * WHERE 1=1")
    or die ("Invalid query");
?>
```

The following query is semantically invalid if *my_col* is not a column in the table *my_tbl*, so **mysql_query()** fails and returns FALSE:

Example 2. mysql_query()

```
<?php
$result = mysql_query ("SELECT my_col FROM my_tbl")
    or die ("Invalid query");
?>
```

mysql_query() will also fail and return FALSE if you don't have permission to access the table(s) referenced by the query.

Assuming the query succeeds, you can call **mysql_num_rows()** to find out how many rows were returned for a SELECT statement or **mysql_affected_rows()** to find out how many rows were affected by a DELETE, INSERT, REPLACE, or UPDATE statement.

For SELECT statements, **mysql_query()** returns a new result identifier that you can pass to **mysql_result()**. When you are done with the result set, you can free the resources associated with it by calling **mysql_free_result()**. Although, the memory will automatically be freed at the end of the script's execution.

See also: **mysql_affected_rows()**, **mysql_db_query()**, **mysql_free_result()**, **mysql_result()**, **mysql_select_db()**, and **mysql_connect()**.

mysql_result (PHP 3, PHP 4)

Get result data

```
mixed mysql_result (int result, int row [, mixed field])
```

mysql_result() returns the contents of one cell from a MySQL result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than **mysql_result()**. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Calls to **mysql_result()** should not be mixed with calls to other functions that deal with the result set.

Recommended high-performance alternatives: **mysql_fetch_row()**, **mysql_fetch_array()**, and **mysql_fetch_object()**.

mysql_select_db (PHP 3, PHP 4)

Select a MySQL database

```
int mysql_select_db (string database_name [, int link_identifier])
```

Returns: true on success, false on error.

mysql_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if **mysql_connect()** was called, and use it.

Every subsequent call to **mysql_query()** will be made on the active database.

See also: **mysql_connect()**, **mysql_pconnect()**, and **mysql_query()**.

For downward compatibility **mysql_selectdb()** can also be used.

mysql_tablename (PHP 3, PHP 4)

Get table name of field

```
string mysql_tablename (int result, int i)
```

mysql_tablename() takes a result pointer returned by the **mysql_list_tables()** function as well as an integer index and returns the name of a table. The **mysql_num_rows()** function may be used to determine the number of tables in the result pointer.

Example 1. Mysql_tablename() Example

```
<?php
mysql_connect ("localhost:3306");
$result = mysql_list_tables ("wisconsin");
$i = 0;
while ($i < mysql_num_rows ($result)) {
    $tb_names[$i] = mysql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```


XLIV. Network Functions

checkdnsrr (PHP 3, PHP 4)

Check DNS records corresponding to a given Internet host name or IP address

```
int checkdnsrr (string host [, string type])
```

Searches DNS for records of type *type* corresponding to *host*. Returns true if any records are found; returns false if no records were found or if an error occurred.

type may be any one of: A, MX, NS, SOA, PTR, CNAME, or ANY. The default is MX.

Host may either be the IP address in dotted-quad notation or the host name.

See also **getmxrr()**, **gethostbyaddr()**, **gethostbyname()**, **gethostbynamel()**, and the named(8) manual page.

closelog (PHP 3, PHP 4)

Close connection to system logger

```
int closelog(void);
```

Closelog() closes the descriptor being used to write to the system logger. The use of **closelog()** is optional.

See also **define_syslog_variables()**, **syslog()** and **openlog()**.

debugger_off (PHP 3)

Disable internal PHP debugger

```
int debugger_off(void);
```

Disables the internal PHP debugger. The debugger is still under development.

debugger_on (PHP 3)

Enable internal PHP debugger

```
int debugger_on (string address)
```

Enables the internal PHP debugger, connecting it to *address*. The debugger is still under development.

define_syslog_variables (PHP 3, PHP 4)

Initializes all syslog related constants

```
void define_syslog_variables (void)
```

Initializes all constants used in the syslog functions.

See also **openlog()**, **syslog()** and **closelog()**.

fsockopen (PHP 3, PHP 4)

Open Internet or Unix domain socket connection

```
int fsockopen (string [udp://]hostname, int port [, int errno [, string errstr [, double timeout]]])
```

Initiates a stream connection in the Internet (AF_INET, using TCP or UDP) or Unix (AF_UNIX) domain. For the Internet domain, it will open a TCP socket connection to *hostname* on port *port*. *hostname* may in this case be either a fully qualified domain name or an IP address. For UDP connections, you need to explicitly specify the protocol: *udp://hostname*. For the Unix domain, *hostname* will be used as the path to the socket, *port* must be set to 0 in this case. The optional *timeout* can be used to set a timeout in seconds for the connect system call.

Fsockopen() returns a file pointer which may be used together with the other file functions (such as **fgets()**, **fgetss()**, **fputs()**, **fclose()**, and **feof()**).

If the call fails, it will return false and if the optional *errno* and *errstr* arguments are present they will be set to indicate the actual system level error that occurred on the system-level **connect()** call. If the returned *errno* is 0 and the function returned false, it is an indication that the error occurred before the **connect()** call. This is most likely due to a problem initializing the socket. Note that the *errno* and *errstr* arguments must be passed by reference.

Depending on the environment, the Unix domain or the optional connect timeout may not be available.

The socket will by default be opened in blocking mode. You can switch it to non-blocking mode by using **socket_set_blocking()**.

Example 1. Fsockopen() Example

```
$fp = fsockopen ("www.php.net", 80, $errno, $errstr, 30);
if (!$fp) {
    echo "$errstr ($errno)<br>\n";
} else {
    fputs ($fp, "GET / HTTP/1.0\r\n\r\n");
    while (!feof($fp)) {
        echo fgets ($fp,128);
    }
    fclose ($fp);
}
```

The example below shows how to retrieve the day and time from the UDP service "daytime" (port 13) in your own machine.

Example 2. Using UDP connection

```
<?php
$fp = fsockopen("udp://127.0.0.1", 13, $errno, $errstr);
if (!$fp) {
    echo "ERROR: $errno - $errstr<br>\n";
} else {
    fwrite($fp, "\n");
    echo fread($fp, 26);
    fclose($fp);
}
?>
```

Note: The timeout parameter was introduced in PHP 3.0.9 and UDP support was added in PHP 4.

See also: **pfsockopen()**, **socket_set_blocking()**, **socket_set_timeout()**, **fgets()**, **fgetss()**, **fputs()**, **fclose()**, and **feof()**.

gethostbyaddr (PHP 3, PHP 4)

Get the Internet host name corresponding to a given IP address

```
string gethostbyaddr (string ip_address)
```

Returns the host name of the Internet host specified by *ip_address*. If an error occurs, returns *ip_address*.

See also **gethostbyname()**.

gethostbyname (PHP 3, PHP 4)

Get the IP address corresponding to a given Internet host name

```
string gethostbyname (string hostname)
```

Returns the IP address of the Internet host specified by *hostname*.

See also **gethostbyaddr()**.

gethostbynamel (PHP 3, PHP 4)

Get a list of IP addresses corresponding to a given Internet host name

```
array gethostbynamel (string hostname)
```

Returns a list of IP addresses to which the Internet host specified by *hostname* resolves.

See also **gethostbyname()**, **gethostbyaddr()**, **checkdnsrr()**, **getmxrr()**, and the `named(8)` manual page.

getmxrr (PHP 3, PHP 4)

Get MX records corresponding to a given Internet host name

```
int getmxrr (string hostname, array mxhosts [, array weight])
```

Searches DNS for MX records corresponding to *hostname*. Returns true if any records are found; returns false if no records were found or if an error occurred.

A list of the MX records found is placed into the array *mxhosts*. If the *weight* array is given, it will be filled with the weight information gathered.

See also **checkdnsrr()**, **gethostbyname()**, **gethostbynamel()**, **gethostbyaddr()**, and the `named(8)` manual page.

getprotobynumber (PHP 4 >= 4.0b4)

Get protocol number associated with protocol name

```
int getprotobynumber (string name)
```

Getprotobynumber() returns the protocol number associated with the protocol *name* as per /etc/protocols.

See also: [getprotobynumber\(\)](#).

getprotobynumber (PHP 4 >= 4.0b4)

Get protocol name associated with protocol number

```
string getprotobynumber (int number)
```

Getprotobynumber() returns the protocol name associated with protocol *number* as per /etc/protocols.

See also: [getprotobynumber\(\)](#).

getservbyname (PHP 4 >= 4.0b4)

Get port number associated with an Internet service and protocol

```
int getservbyname (string service, string protocol)
```

Getservbyname() returns the Internet port which corresponds to *service* for the specified *protocol* as per /etc/services. *protocol* is either TCP or UDP.

See also: [getservbyport\(\)](#).

getservbyport (PHP 4 >= 4.0b4)

Get Internet service which corresponds to port and protocol

```
string getservbyport (int port, string protocol)
```

Getservbyport() returns the Internet service associated with *port* for the specified *protocol* as per /etc/services. *protocol* is either TCP or UDP.

See also: [getservbyname\(\)](#).

ip2long (PHP 4 >= 4.0RC1)

Converts a string containing an (IPv4) Internet Protocol dotted address into a proper address.

```
int ip2long (string ip_address)
```

The function **ip2long()** generates an IPv4 Internet network address from its Internet standard format (dotted string) representation.

Example 1. Ip2long() Example

```
<?php
$ip = gethostbyname( "www.php.net" );
$out = "The following URLs are equivalent:<br>\n";
$out .= "http://www.php.net/, http://".$ip."/ , and http://".ip2long($ip)."/<br>\n";
echo $out;
?>
```

See also: **long2ip()**

long2ip (PHP 4 >= 4.0RC1)

Converts an (IPv4) Internet network address into a string in Internet standard dotted format

```
string long2ip ( int proper_address )
```

The function **long2ip()** generates an Internet address in dotted format (i.e.: aaa.bbb.ccc.ddd) from the proper address representation.

See also: **ip2long()**

openlog (PHP 3, PHP 4)

Open connection to system logger

```
int openlog (string ident, int option, int facility)
```

Openlog() opens a connection to the system logger for a program. The string *ident* is added to each message. Values for *option* and *facility* are given below. The *option* argument is used to indicate what loggin options will be used when generating a log message. The *facility* argument is used to specify what type of program is logging the message. This allows you to specify (in your machine's syslog configuration) how messages coming from different facilities will be handled. The use of **openlog()** is optional. It will automatically be called by **syslog()** if necessary, in which case *ident* will default to false.

Table 1. Openlog() Options

Constant	Description
LOG_CONS	if there is an error while sending data to the system logger, write directly to the system console
LOG_NDELAY	open the connection to the logger immediately
LOG_ODELAY	(default) delay openning the connection until the first message is logged
LOG_PERROR	print log message also to standard error
LOG_PID	include PID with each message

You can use one or more of this options. When using multiple options you need to OR them, i.e. to open the connection

immediately, write to the console and include the PID in each message, you will use: `LOG_CONS | LOG_NDELAY | LOG_PID`

Table 2. Openlog() Facilities

Constant	Description
<code>LOG_AUTH</code>	security/authorization messages (use <code>LOG_AUTHPRIV</code> instead in systems where that constant is defined)
<code>LOG_AUTHPRIV</code>	security/authorization messages (private)
<code>LOG_CRON</code>	clock daemon (cron and at)
<code>LOG_DAEMON</code>	other system daemons
<code>LOG_KERN</code>	kernel messages
<code>LOG_LOCAL0 ... LOG_LOCAL7</code>	reserved for local use
<code>LOG_LPR</code>	line printer subsystem
<code>LOG_MAIL</code>	mail subsystem
<code>LOG_NEWS</code>	USENET news subsystem
<code>LOG_SYSLOG</code>	messages generated internally by syslogd
<code>LOG_USER</code>	generic user-level messages
<code>LOG_UUCP</code>	UUCP subsystem

See also `define_syslog_variables()`, `syslog()` and `closelog()`.

pfsockopen (PHP 3>= 3.0.7, PHP 4)

Open persistent Internet or Unix domain socket connection

```
int pfsockopen (string hostname, int port [, int errno [, string errstr [, int timeout]]])
```

This function behaves exactly as `fsockopen()` with the difference that the connection is not closed after the script finishes. It is the persistent version of `fsockopen()`.

socket_get_status (PHP 4 >= 4.0b4)

Returns information about existing socket resource

```
array socket_get_status (resource socket_get_status)
```

Returns information about an existing socket resource. Currently returns four entries in the result array:

- `timed_out` (bool) - The socket timed out waiting for data
- `blocked` (bool) - The socket was blocked
- `eof` (bool) - Indicates EOF event
- `unread_bytes` (int) - Number of bytes left in the socket buffer

See also `accept_connect()`, `bind()`, `connect()`, `listen()`, and `strerror()`.

socket_set_blocking (PHP 4 >= 4.0b4)

Set blocking/non-blocking mode on a socket

```
int socket_set_blocking (int socket descriptor, int mode)
```

If *mode* is false, the given socket descriptor will be switched to non-blocking mode, and if true, it will be switched to blocking mode. This affects calls like **fgets()** that read from the socket. In non-blocking mode an **fgets()** call will always return right away while in blocking mode it will wait for data to become available on the socket.

This function was previously called as **set_socket_blocking()** but this usage is deprecated.

socket_set_timeout (PHP 4 >= 4.0b4)

Set timeout period on a socket

```
bool socket_set_timeout (int socket descriptor, int seconds, int microseconds)
```

Sets the timeout value on *socket descriptor*, expressed in the sum of *seconds* and *microseconds*.

Example 1. socket_set_timeout() Example

```
<?php
$fp = fsockopen("www.php.net", 80);
if(!$fp) {
    echo "Unable to open\n";
} else {
    fputs($fp, "GET / HTTP/1.0\n\n");
    $start = time();
    socket_set_timeout($fp, 2);
    $res = fread($fp, 2000);
    var_dump(socket_get_status($fp));
    fclose($fp);
    print $res;
}
?>
```

This function was previously called as **set_socket_timeout()** but this usage is deprecated.

See also: **fsockopen()** and **fopen()**.

syslog (PHP 3, PHP 4)

Generate a system log message

```
int syslog (int priority, string message)
```

Syslog() generates a log message that will be distributed by the system logger. *priority* is a combination of the facility and the level, values for which are given in the next section. The remaining argument is the message to send, except that the two characters %m will be replaced by the error message string (strerror) corresponding to the present value of errno.

Table 1. Syslog() Priorities (in descending order)

Constant	Description
LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal, but significant, condition
LOG_INFO	informational message
LOG_DEBUG	debug-level message

Example 1. Using syslog()

```
<?php
define_syslog_variables();
// open syslog, include the process ID and also send
// the log to standard error, and use a user defined
// logging mechanism
openlog("myScripLog", LOG_PID | LOG_PERROR, LOG_LOCAL0);

// some code

if (authorized_client()) {
    // do something
} else {
    // unauthorized client!
    // log the attempt
    $access = date("Y/m/d H:i:s");
    syslog(LOG_WARNING, "Unauthorized client: $access $REMOTE_ADDR ($HTTP_USER_AGENT)");
}

closelog();
?>
```

For information on setting up a user defined log handler, see the `syslog.conf(5)` Unix manual page. More information on the syslog facilities and option can be found in the man pages for `syslog(3)` on Unix machines.

On Windows NT, the syslog service is emulated using the Event Log.

See also `define_syslog_variables()`, `openlog()` and `closelog()`.

XLV. Unified ODBC functions

In addition to normal ODBC support, the Unified ODBC functions in PHP allow you to access several databases that have borrowed the semantics of the ODBC API to implement their own API. Instead of maintaining multiple database drivers that were all nearly identical, these drivers have been unified into a single set of ODBC functions.

The following databases are supported by the Unified ODBC functions: Adabas D (<http://www.adabas.com/>), IBM DB2 (<http://www.ibm.com/db2/>), iODBC (<http://www.iodbc.org/>), Solid (<http://www.solidtech.com/>), and Sybase SQL Anywhere (<http://www.sybase.com/>).

Note: There is no ODBC involved when connecting to the above databases. The functions that you use to speak natively to them just happen to share the same names and syntax as the ODBC functions.

odbc_autocommit (PHP 3>= 3.0.6, PHP 4)

Toggle autocommit behaviour

```
int odbc_autocommit (int connection_id [, int OnOff])
```

Without the *OnOff* parameter, this function returns auto-commit status for *connection_id*. True is returned if auto-commit is on, false if it is off or an error occurs.

If *OnOff* is true, auto-commit is enabled, if it is false auto-commit is disabled. Returns `true` on success, `false` on failure.

By default, auto-commit is on for a connection. Disabling auto-commit is equivalent with starting a transaction.

See also **odbc_commit()** and **odbc_rollback()**.

odbc_binmode (PHP 3>= 3.0.6, PHP 4)

Handling of binary column data

```
int odbc_binmode (int result_id, int mode)
```

(ODBC SQL types affected: BINARY, VARBINARY, LONGVARBINARY)

- ODBC_BINMODE_PASSTHRU: Passthru BINARY data
- ODBC_BINMODE_RETURN: Return as is
- ODBC_BINMODE_CONVERT: Convert to char and return

When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to "01" and a binary 11111111 is converted to "FF".

Table 1. LONGVARBINARY handling

binmode	longreadlen	result
ODBC_BINMODE_PASSTHRU	0	passthru
ODBC_BINMODE_RETURN	0	passthru
ODBC_BINMODE_CONVERT	0	passthru
ODBC_BINMODE_PASSTHRU	0	passthru
ODBC_BINMODE_PASSTHRU	>0	passthru
ODBC_BINMODE_RETURN	>0	return as is
ODBC_BINMODE_CONVERT	>0	return as char

If **odbc_fetch_into()** is used, passthru means that an empty string is returned for these columns.

If *result_id* is 0, the settings apply as default for new results.

Note: Default for longreadlen is 4096 and binmode defaults to ODBC_BINMODE_RETURN. Handling of binary long columns is also affected by **odbc_longreadlen()**

odbc_close (PHP 3>= 3.0.6, PHP 4)

Close an ODBC connection

```
void odbc_close (int connection_id)
```

odbc_close() will close down the connection to the database server associated with the given connection identifier.

Note: This function will fail if there are open transactions on this connection. The connection will remain open in this case.

odbc_close_all (PHP 3>= 3.0.6, PHP 4)

Close all ODBC connections

```
void odbc_close_all(void);
```

odbc_close_all() will close down all connections to database server(s).

Note: This function will fail if there are open transactions on a connection. This connection will remain open in this case.

odbc_commit (PHP 3>= 3.0.6, PHP 4)

Commit an ODBC transaction

```
int odbc_commit (int connection_id)
```

Returns: `true` on success, `false` on failure. All pending transactions on *connection_id* are committed.

odbc_connect (PHP 3>= 3.0.6, PHP 4)

Connect to a datasource

```
int odbc_connect (string dsn, string user, string password [, int cursor_type])
```

Returns an ODBC connection id or 0 (`false`) on error.

The connection id returned by this functions is needed by other ODBC functions. You can have multiple connections open at once. The optional fourth parameter sets the type of cursor to be used for this connection. This parameter is not normally needed, but can be useful for working around problems with some ODBC drivers.

With some ODBC drivers, executing a complex stored procedure may fail with an error similar to: "Cannot open a cursor on a stored procedure that has anything other than a single select statement in it". Using `SQL_CUR_USE_ODBC` may

avoid that error. Also, some drivers don't support the optional row_number parameter in **odbc_fetch_row()**. SQL_CUR_USE_ODBC might help in that case, too.

The following constants are defined for cursortype:

- SQL_CUR_USE_IF_NEEDED
- SQL_CUR_USE_ODBC
- SQL_CUR_USE_DRIVER
- SQL_CUR_DEFAULT

For persistent connections see **odbc_pconnect()**.

odbc_cursor (PHP 3>= 3.0.6, PHP 4)

Get cursorname

```
string odbc_cursor (int result_id)
```

odbc_cursor will return a cursorname for the given result_id.

odbc_do (PHP 3>= 3.0.6, PHP 4)

Synonym for **odbc_exec()**

```
int odbc_do (int conn_id, string query)
```

Odbc_do() will execute a query on the given connection.

odbc_error (unknown)

Get the last error code

```
int odbc_error ([int connection_id])
```

Returns a six-digit ODBC state, or an empty string if there has been no errors. If *connection_id* is specified, the last state of that connection is returned, else the last state of any connection is returned.

See also: **odbc_errormsg()** and **odbc_exec()**.

odbc_errormsg (unknown)

Get the last error message

```
int odbc_errormsg ([int connection_id])
```

Returns a string containing the last ODBC error message, or an empty string if there has been no errors. If *connection_id* is specified, the last state of that connection is returned, else the last state of any connection is returned.
 See also: **odbc_error()** and **odbc_exec()**.

odbc_exec (PHP 3>= 3.0.6, PHP 4)

Prepare and execute a SQL statement

```
int odbc_exec (int connection_id, string query_string)
```

Returns `false` on error. Returns an ODBC result identifier if the SQL command was executed successfully.

odbc_exec() will send an SQL statement to the database server specified by *connection_id*. This parameter must be a valid identifier returned by **odbc_connect()** or **odbc_pconnect()**.

See also: **odbc_prepare()** and **odbc_execute()** for multiple execution of SQL statements.

odbc_execute (PHP 3>= 3.0.6, PHP 4)

Execute a prepared statement

```
int odbc_execute (int result_id [, array parameters_array])
```

Executes a statement prepared with **odbc_prepare()**. Returns `true` on successful execution, `false` otherwise. The array *parameters_array* only needs to be given if you really have parameters in your statement.

odbc_fetch_into (PHP 3>= 3.0.6, PHP 4)

Fetch one result row into array

```
int odbc_fetch_into (int result_id [, int rownumber, array result_array])
```

Returns the number of columns in the result; `false` on error. *result_array* must be passed by reference, but it can be of any type since it will be converted to type array. The array will contain the column values starting at array index 0.

odbc_fetch_row (PHP 3>= 3.0.6, PHP 4)

Fetch a row

```
int odbc_fetch_row (int result_id [, int row_number])
```

If **odbc_fetch_row()** was successful (there was a row), `true` is returned. If there are no more rows, `false` is returned.

odbc_fetch_row() fetches a row of the data that was returned by **odbc_do()** / **odbc_exec()**. After **odbc_fetch_row()** is called, the fields of that row can be accessed with **odbc_result()**.

If *row_number* is not specified, **odbc_fetch_row()** will try to fetch the next row in the result set. Calls to **odbc_fetch_row()** with and without *row_number* can be mixed.

To step through the result more than once, you can call **odbc_fetch_row()** with *row_number* 1, and then continue doing **odbc_fetch_row()** without *row_number* to review the result. If a driver doesn't support fetching rows by number, the *row_number* parameter is ignored.

odbc_field_name (PHP 3>= 3.0.6, PHP 4)

Get the columnname

```
string odbc_field_name ( int result_id, int field_number )
```

odbc_field_name() will return the name of the field occupying the given column number in the given ODBC result identifier. Field numbering starts at 1. `false` is returned on error.

odbc_field_num (PHP 3>= 3.0.6, PHP 4)

Return column number

```
int odbc_field_num ( int result_id, string field_name )
```

odbc_field_num() will return the number of the column slot that corresponds to the named field in the given ODBC result identifier. Field numbering starts at 1. `false` is returned on error.

odbc_field_type (PHP 3>= 3.0.6, PHP 4)

Datatype of a field

```
string odbc_field_type ( int result_id, int field_number )
```

odbc_field_type() will return the SQL type of the field referecend by number in the given ODBC result identifier. Field numbering starts at 1.

odbc_field_len (PHP 3>= 3.0.6, PHP 4)

Get the length (precision) of a field

```
int odbc_field_len ( int result_id, int field_number )
```

odbc_field_len() will return the length of the field referecend by number in the given ODBC result identifier. Field numbering starts at 1.

See also: **odbc_field_scale()** to get the scale of a floating point number.

odbc_field_precision (PHP 4 >= 4.0.0)

Synonym for **odbc_field_len()**

```
string odbc_field_precision (int result_id, int field_number)
```

odbc_field_precision() will return the precision of the field referenced by number in the given ODBC result identifier.

See also: **odbc_field_scale()** to get the scale of a floating point number.

odbc_field_scale (PHP 4 >= 4.0.0)

Get the scale of a field

```
string odbc_field_scale (int result_id, int field_number)
```

odbc_field_precision() will return the scale of the field referenced by number in the given ODBC result identifier.

odbc_free_result (PHP 3>= 3.0.6, PHP 4)

Free resources associated with a result

```
int odbc_free_result (int result_id)
```

Always returns true.

odbc_free_result() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call **odbc_free_result()**, and the memory associated with *result_id* will be freed.

Note: If auto-commit is disabled (see **odbc_autocommit()**) and you call **odbc_free_result()** before committing, all pending transactions are rolled back.

odbc_longreadlen (PHP 3>= 3.0.6, PHP 4)

Handling of LONG columns

```
int odbc_longreadlen (int result_id, int length)
```

(ODBC SQL types affected: LONG, LONGVARBINARY) The number of bytes returned to PHP is controlled by the parameter length. If it is set to 0, Long column data is passed thru to the client.

Note: Handling of LONGVARBINARY columns is also affected by **odbc_binmode()**.

odbc_num_fields (PHP 3>= 3.0.6, PHP 4)

Number of columns in a result

```
int odbc_num_fields (int result_id)
```

Odbc_num_fields() will return the number of fields (columns) in an ODBC result. This function will return -1 on error. The argument is a valid result identifier returned by **odbc_exec()**.

odbc_pconnect (PHP 3>= 3.0.6, PHP 4)

Open a persistent database connection

```
int odbc_pconnect (string dsn, string user, string password [, int cursor_type])
```

Returns an ODBC connection id or 0 (`false`) on error. This function is much like **odbc_connect()**, except that the connection is not really closed when the script has finished. Future requests for a connection with the same `dsn`, `user`, `password` combination (via **odbc_connect()** and **odbc_pconnect()**) can reuse the persistent connection.

Note: Persistent connections have no effect if PHP is used as a CGI program.

For information about the optional `cursor_type` parameter see the **odbc_connect()** function. For more information on persistent connections, refer to the PHP FAQ.

odbc_prepare (PHP 3>= 3.0.6, PHP 4)

Prepares a statement for execution

```
int odbc_prepare (int connection_id, string query_string)
```

Returns `false` on error.

Returns an ODBC result identifier if the SQL command was prepared successfully. The result identifier can be used later to execute the statement with **odbc_execute()**.

odbc_num_rows (PHP 3>= 3.0.6, PHP 4)

Number of rows in a result

```
int odbc_num_rows (int result_id)
```

odbc_num_rows() will return the number of rows in an ODBC result. This function will return -1 on error. For INSERT, UPDATE and DELETE statements **odbc_num_rows()** returns the number of rows affected. For a SELECT clause this can be the number of rows available.

Note: Using **odbc_num_rows()** to determine the number of rows available after a SELECT will return -1 with many drivers.

odbc_result (PHP 3>= 3.0.6, PHP 4)

Get result data

```
string odbc_result (int result_id, mixed field)
```

Returns the contents of the field.

field can either be an integer containing the column number of the field you want; or it can be a string containing the name of the field. For example:

```
$item_3 = odbc_result ($Query_ID, 3);
$item_val = odbc_result ($Query_ID, "val");
```

The first call to **odbc_result()** returns the value of the third field in the current record of the query result. The second function call to **odbc_result()** returns the value of the field whose field name is "val" in the current record of the query result. An error occurs if a column number parameter for a field is less than one or exceeds the number of columns (or fields) in the current record. Similarly, an error occurs if a field with a name that is not one of the fieldnames of the table(s) that is(are) being queried.

Field indices start from 1. Regarding the way binary or long column data is returned refer to **odbc_binmode()** and **odbc_longreadlen()**.

odbc_result_all (PHP 3>= 3.0.6, PHP 4)

Print result as HTML table

```
int odbc_result_all (int result_id [, string format])
```

Returns the number of rows in the result or false on error.

OdBC_result_all() will print all rows from a result identifier produced by **odbc_exec()**. The result is printed in HTML table format. With the optional string argument *format*, additional overall table formatting can be done.

odbc_rollback (PHP 3>= 3.0.6, PHP 4)

Rollback a transaction

```
int odbc_rollback (int connection_id)
```

Rolls back all pending statements on *connection_id*. Returns true on success, false on failure.

odbc_setoption (PHP 3>= 3.0.6, PHP 4)

Adjust ODBC settings. Returns false if an error occurs, otherwise true.

```
int odbc_setoption (int id, int function, int option, int param)
```

This function allows fiddling with the ODBC options for a particular connection or query result. It was written to help find workarounds to problems in quirky ODBC drivers. You should probably only use this function if you are an ODBC programmer and understand the effects the various options will have. You will certainly need a good ODBC reference to explain all the different options and values that can be used. Different driver versions support different options.

Because the effects may vary depending on the ODBC driver, use of this function in scripts to be made publicly available is strongly discouraged. Also, some ODBC options are not available to this function because they must be set before the connection is established or the query is prepared. However, if on a particular job it can make PHP work so your boss doesn't tell you to use a commercial product, that's all that really matters.

ID is a connection id or result id on which to change the settings. For `SQLSetConnectOption()`, this is a connection id. For `SQLSetStmtOption()`, this is a result id.

Function is the ODBC function to use. The value should be 1 for `SQLSetConnectOption()` and 2 for `SQLSetStmtOption()`.

Parameter *option* is the option to set.

Parameter *param* is the value for the given *option*.

Example 1. ODBC Setoption Examples

```
// 1. Option 102 of SQLSetConnectOption() is SQL_AUTOCOMMIT.
//     Value 1 of SQL_AUTOCOMMIT is SQL_AUTOCOMMIT_ON.
//     This example has the same effect as
//     odbc_autocommit($conn, true);

odbc_setoption ($conn, 1, 102, 1);

// 2. Option 0 of SQLSetStmtOption() is SQL_QUERY_TIMEOUT.
//     This example sets the query to timeout after 30 seconds.

$result = odbc_prepare ($conn, $sql);
odbc_setoption ($result, 2, 0, 30);
odbc_execute ($result);
```

odbc_tables (PHP 3>= 3.0.17, PHP 4 >= 4.0b4)

Get the list of table names stored in a specific data source. Returns a result identifier containing the information.

```
int odbc_tables (int connection_id [, string qualifier [, string owner [, string name [, string types]]]])
```

Lists all tables in the requested range. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- TABLE_TYPE
- REMARKS

The result set is ordered by TABLE_TYPE, TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

The *owner* and *name* arguments accept search patterns ('%' to match zero or more characters and '_' to match a single character).

To support enumeration of qualifiers, owners, and table types, the following special semantics for the *qualifier*, *owner*, *name*, and *table_type* are available:

- If *qualifier* is a single percent character (%) and *owner* and *name* are empty strings, then the result set contains a list of valid qualifiers for the data source. (All columns except the TABLE_QUALIFIER column contain NULLs.)
- If *owner* is a single percent character (%) and *qualifier* and *name* are empty strings, then the result set contains a list of valid owners for the data source. (All columns except the TABLE_OWNER column contain NULLs.)
- If *table_type* is a single percent character (%) and *qualifier*, *owner* and *name* are empty strings, then the result set contains a list of valid table types for the data source. (All columns except the TABLE_TYPE column contain NULLs.)

If *table_type* is not an empty string, it must contain a list of comma-separated values for the types of interest; each value may be enclosed in single quotes ('') or unquoted. For example, "'TABLE','VIEW'" or "TABLE, VIEW". If the data source does not support a specified table type, **odbc_tables()** does not return any results for that type.

See also **odbc_tableprivileges()** to retrieve associated privileges.

odbc_tableprivileges (PHP 4 >= 4.0b4)

Lists tables and the privileges associated with each table

```
int odbc_tableprivileges (int connection_id [, string qualifier [, string owner [, string name]]])
```

Lists tables in the requested range and the privileges associated with each table. Returns an ODBC result identifier or **false** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- GRANTOR
- GRANTEE
- PRIVILEGE
- IS_GRANTABLE

The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

The *owner* and *name* arguments accept search patterns ('%' to match zero or more characters and '_' to match a single character).

odbc_columns (PHP 4 >= 4.0b4)

Lists the column names in specified tables. Returns a result identifier containing the information.

```
int odbc_columns (int connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]])
```

Lists all columns in the requested range. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- COLUMN_NAME
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE
- RADIX
- NULLABLE
- REMARKS

The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

The *owner*, *table_name* and *column_name* arguments accept search patterns ('%' to match zero or more characters and '_' to match a single character).

See also **odbc_columnprivileges()** to retrieve associated privileges.

odbc_columnprivileges (PHP 4 >= 4.0b4)

Returns a result identifier that can be used to fetch a list of columns and associated privileges

```
int odbc_columnprivileges (int connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]])
```

Lists columns and associated privileges for the given table. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- GRANTOR
- GRANTEE

- PRIVILEGE
- IS_GRANTABLE

The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

The *column_name* argument accepts search patterns ('%' to match zero or more characters and '_' to match a single character).

odbc_gettypeinfo (PHP 4 >= 4.0b4)

Returns a result identifier containing information about data types supported by the data source.

```
int odbc_gettypeinfo (int connection_id [, int data_type])
```

Retrieves information about data types supported by the data source. Returns an ODBC result identifier or false on failure. The optional argument *data_type* can be used to restrict the information to a single data type.

The result set has the following columns:

- TYPE_NAME
- DATA_TYPE
- PRECISION
- LITERAL_PREFIX
- LITERAL_SUFFIX
- CREATE_PARAMS
- NULLABLE
- CASE_SENSITIVE
- SEARCHABLE
- UNSIGNED_ATTRIBUTE
- MONEY
- AUTO_INCREMENT
- LOCAL_TYPE_NAME
- MINIMUM_SCALE
- MAXIMUM_SCALE

The result set is ordered by DATA_TYPE and TYPE_NAME.

odbc_primarykeys (PHP 4 >= 4.0b4)

Returns a result identifier that can be used to fetch the column names that comprise the primary key for a table

```
int odbc_primarykeys (int connection_id, string qualifier, string owner, string table)
```

Returns the column names that comprise the primary key for a table. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- COLUMN_NAME
- KEY_SEQ
- PK_NAME

odbc_foreignkeys (PHP 4 >= 4.0b4)

Returns a list of foreign keys in the specified table or a list of foreign keys in other tables that refer to the primary key in the specified table

```
int odbc_foreignkeys (int connection_id, string pk_qualifier, string pk_owner, string
pk_table, string fk_qualifier, string fk_owner, string fk_table)
```

Odbc_foreignkeys() retrieves information about foreign keys. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- PKTABLE_QUALIFIER
- PKTABLE_OWNER
- PKTABLE_NAME
- PKCOLUMN_NAME
- FKTABLE_QUALIFIER
- FKTABLE_OWNER
- FKTABLE_NAME
- FKCOLUMN_NAME
- KEY_SEQ
- UPDATE_RULE
- DELETE_RULE
- FK_NAME
- PK_NAME

If `pk_table` contains a table name, **odbc_foreignkeys()** returns a result set containing the primary key of the specified table and all of the foreign keys that refer to it.

If `fk_table` contains a table name, **odbc_foreignkeys()** returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both `pk_table` and `fk_table` contain table names, **odbc_foreignkeys()** returns the foreign keys in the table specified in `fk_table` that refer to the primary key of the table specified in `pk_table`. This should be one key at most.

odbc_procedures (PHP 4 >= 4.0b4)

Get the list of procedures stored in a specific data source. Returns a result identifier containing the information.

```
int odbc_procedures (int connection_id [, string qualifier [, string owner [, string name] ]])
```

Lists all procedures in the requested range. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- NUM_INPUT_PARAMS
- NUM_OUTPUT_PARAMS
- NUM_RESULT_SETS
- REMARKS
- PROCEDURE_TYPE

The *owner* and *name* arguments accept search patterns ('%' to match zero or more characters and '_' to match a single character).

odbc_procedurecolumns (PHP 4 >= 4.0b4)

Retrieve information about parameters to procedures

```
int odbc_procedurecolumns (int connection_id [, string qualifier [, string owner [, string proc [, string column] ]]])
```

Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- COLUMN_NAME
- COLUMN_TYPE
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE

- RADIX
- NULLABLE
- REMARKS

The result set is ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME and COLUMN_TYPE.

The *owner*, *proc* and *column* arguments accept search patterns ('%' to match zero or more characters and '_' to match a single character).

odbc_specialcolumns (PHP 4 >= 4.0b4)

Returns either the optimal set of columns that uniquely identifies a row in the table or columns that are automatically updated when any value in the row is updated by a transaction

```
int odbc_specialcolumns (int connection_id, int type, string qualifier, string owner,
string table, int scope, int nullable)
```

When the type argument is SQL_BEST_ROWID, **odbc_specialcolumns()** returns the column or columns that uniquely identify each row in the table.

When the type argument is SQL_ROWVER, **odbc_specialcolumns()** returns the optimal column or set of columns that, by retrieving values from the column or columns, allows any row in the specified table to be uniquely identified.

Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- SCOPE
- COLUMN_NAME
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE
- PSEUDO_COLUMN

The result set is ordered by SCOPE.

odbc_statistics (PHP 4 >= 4.0b4)

Retrieve statistics about a table

```
int odbc_statistics (int connection_id, string qualifier, string owner, string table_name,
int unique, int accuracy)
```

Get statistics about a table and it's indexes. Returns an ODBC result identifier or `false` on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- NON_UNIQUE
- INDEX_QUALIFIER
- INDEX_NAME
- TYPE
- SEQ_IN_INDEX
- COLUMN_NAME
- COLLATION
- CARDINALITY
- PAGES
- FILTER_CONDITION

The result set is ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME and SEQ_IN_INDEX.

XLVI. Oracle 8 functions

These functions allow you to access Oracle8 and Oracle7 databases. It uses the Oracle8 Call-Interface (OCI8). You will need the Oracle8 client libraries to use this extension.

This extension is more flexible than the standard Oracle extension. It supports binding of global and local PHP variables to Oracle placeholders, has full LOB, FILE and ROWID support and allows you to use user-supplied define variables.

Before using this extension, make sure that you have set up your oracle environment variables properly for the Oracle user, as well as your web daemon user. The variables you might need to set are as follows:

- ORACLE_HOME
- ORACLE_SID
- LD_PRELOAD
- LD_LIBRARY_PATH
- NLS_LANG
- ORA_NLS33

After setting up the environment variables for your webserver user, be sure to also add the webserver user (nobody, www) to the oracle group.

Example 1. OCI Hints

```
<?php
// by sergo@bacup.ru

// Use option: OCI_DEFAULT for execute command to delay execution
OCIEexecute($stmt, OCI_DEFAULT);

// for retrieve data use (after fetch):

$result = OCIResult($stmt, $n);
if (is_object ($result)) $result = $result->load();

// For INSERT or UPDATE statement use:

$sql = "insert into table (field1, field2) values (field1 = 'value',
    field2 = empty_clob()) returning field2 into :field2";
OCIParse($conn, $sql);
$clob = OCINewDescriptor($conn, OCI_D_LOB);
OCIBindByName ($stmt, ":field2", &$clob, -1, OCI_B_CLOB);
OCIEexecute($stmt, OCI_DEFAULT);
$clob->save ("some text");

?>
```

You can easily access stored procedures in the same way as you would from the commands line.

Example 2. Using Stored Procedures

```
<?php
// by webmaster@remoterealty.com
$sth = OCIParse ( $dbh, "begin sp_newaddress( :address_id, '$firstname',
    '$lastname', '$company', '$address1', '$address2', '$city', '$state',
```

```
'$postalcode', '$country', :error_code );end;" );  
  
// This calls stored procedure sp_newaddress, with :address_id being an  
// in/out variable and :error_code being an out variable.  
// Then you do the binding:  
  
OCIBindByName ( $sth, ":address_id", $addr_id, 10 );  
OCIBindByName ( $sth, ":error_code", $errorcode, 10 );  
OCIEexecute ( $sth );  
  
?>
```

OCIDefineByName (PHP 3>= 3.0.7, PHP 4)

Use a PHP variable for the define-step during a SELECT

```
int OCIDefineByName (int stmt, string Column-Name, mixed variable [, int type])
```

OCIDefineByName() uses fetches SQL-Columns into user-defined PHP-Variables. Be careful that Oracle user ALL-UPPERCASE column-names, whereby in your select you can also write lower-case. **OCIDefineByName()** expects the *Column-Name* to be in uppercase. If you define a variable that doesn't exists in you select statement, no error will be given!

If you need to define an abstract Datatype (LOB/ROWID/BFILE) you need to allocate it first using **OCINewDescriptor()** function. See also the **OCIBindByName()** function.

Example 1. OCIDefineByName

```
<?php
/* OCIDefineByPos example thies@thieso.net (980219) */

$conn = OCILogon("scott","tiger");

$stmt = OCIParse($conn,"select empno, ename from emp");

/* the define MUST be done BEFORE ociexecute! */

OCIDefineByName($stmt,"EMPNO",$empno);
OCIDefineByName($stmt,"ENAME",$ename);

OCIEexecute($stmt);

while (OCIFetch($stmt)) {
    echo "empno:".$empno."\n";
    echo "ename:".$ename."\n";
}

OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIBindByName (PHP 3>= 3.0.4, PHP 4)

Bind a PHP variable to an Oracle Placeholder

```
int OCIBindByName (int stmt, string ph_name, mixed &variable, int length [, int type])
```

OCIBindByName() binds the PHP variable *variable* to the Oracle placeholder *ph_name*. Whether it will be used for input or output will be determined run-time, and the necessary storage space will be allocated. The *length* parameter sets the maximum length for the bind. If you set *length* to -1 **OCIBindByName()** will use the current length of *variable* to set the maximum length.

If you need to bind an abstract Datatype (LOB/ROWID/BFILE) you need to allocate it first using **OCINewDescriptor()** function. The *length* is not used for abstract Datatypes and should be set to -1. The *type* variable tells oracle, what kind of descriptor we want to use. Possible values are: OCI_B_FILE (Binary-File), OCI_B_CFILE (Character-File), OCI_B_CLOB (Character-LOB), OCI_B_BLOB (Binary-LOB) and OCI_B_ROWID (ROWID).

Example 1. OCIDefineByName

```
<?php
/* OCIBindByPos example thies@thieso.net (980221)
   inserts 3 records into emp, and uses the ROWID for updating the
   records just after the insert.
*/
$conn = OCILogon("scott","tiger");

$stmt = OCIParse($conn,"insert into emp (empno, ename) ".
    "values (:empno,:ename) ".
    "returning ROWID into :rid");

$data = array(1111 => "Larry", 2222 => "Bill", 3333 => "Jim");

$rowid = OCINewDescriptor($conn,OCI_D_ROWID);

OCIBindByName($stmt,:empno,&$empno,32);
OCIBindByName($stmt,:ename,&$ename,32);
OCIBindByName($stmt,:rid,&$rowid,-1,OCI_B_ROWID);

$update = OCIParse($conn,"update emp set sal = :sal where ROWID = :rid");
OCIBindByName($update,:rid,&$rowid,-1,OCI_B_ROWID);
OCIBindByName($update,:sal,&$sal,32);

$sal = 10000;

while (list($empno,$ename) = each($data)) {
    OCIEexecute($stmt);
    OCIEexecute($update);
}

$rowid->free();

OCIFreeStatement($update);
OCIFreeStatement($stmt);

$stmt = OCIParse($conn,"select * from emp where empno in (1111,2222,3333)");
OCIEexecute($stmt);
while (OCIFetchInto($stmt,&$arr,OCI_ASSOC)) {
    var_dump($arr);
}
OCIFreeStatement($stmt);

/* delete our "junk" from the emp table.... */
$stmt = OCIParse($conn,"delete from emp where empno in (1111,2222,3333)");
OCIEexecute($stmt);
OCIFreeStatement($stmt);

OCILogoff($conn);
?>
```

Warning

It is a bad idea to use magic quotes and **OciBindByName()** simultaneously as no quoting is needed on quoted variables and any quotes magically applied will be written into your database as **OciBindByName()** is not able to distinguish magically added quotings from those added by intention.

OCILogon (PHP 3>= 3.0.4, PHP 4)

Establishes a connection to Oracle

```
int OCILogon (string username, string password [, string db])
```

OCILogon() returns an connection identifier needed for most other OCI calls. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

Connections are shared at the page level when using **OCILogon()**. This means that commits and rollbacks apply to all open transactions in the page, even if you have created multiple connections.

This example demonstrates how the connections are shared.

Example 1. OCILogon

```
<?php
print "<HTML><PRE>" ;
$db = "";

$c1 = ocilogon("scott","tiger",$db);
$c2 = ocilogon("scott","tiger",$db);

function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test varchar2(64))");
  ociexecute($stmt);
  echo $conn." created table\n\n";
}

function drop_table($conn)
{ $stmt = ociparse($conn,"drop table scott.hallo");
  ociexecute($stmt);
  echo $conn." dropped table\n\n";
}

function insert_data($conn)
{ $stmt = ociparse($conn,"insert into scott.hallo
      values('$conn' || ' ' || to_char(sysdate,'DD-MON-YY HH24:MI:SS'))");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." inserted hallo\n\n";
}

function delete_data($conn)
{ $stmt = ociparse($conn,"delete from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." deleted hallo\n\n";
}

function commit($conn)
{ ocicommit($conn);
  echo $conn." committed\n\n";
}

function rollback($conn)
{ ocirollback($conn);
  echo $conn." rollback\n\n";
}
```

```

function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."---selecting\n\n";
  while (ocifetch($stmt))
    echo $conn." <".ociresult($stmt,"TEST").">\n\n";
  echo $conn."---done\n\n";
}

create_table($c1);
insert_data($c1); // Insert a row using c1
insert_data($c2); // Insert a row using c2

select_data($c1); // Results of both inserts are returned
select_data($c2);

rollback($c1); // Rollback using c1

select_data($c1); // Both inserts have been rolled back
select_data($c2);

insert_data($c2); // Insert a row using c2
commit($c2); // commit using c2

select_data($c1); // result of c2 insert is returned

delete_data($c1); // delete all rows in table using c1
select_data($c1); // no rows returned
select_data($c2); // no rows returned
commit($c1); // commit using c1

select_data($c1); // no rows returned
select_data($c2); // no rows returned

drop_table($c1);
print "</PRE></HTML>";
?>

```

See also **OCIPLogon()** and **OCINLogon()**.

OCIPLogon (PHP 3>= 3.0.8, PHP 4)

Connect to an Oracle database and log on using a persistant connection. Returns a new session.

```
int OCIPLogon (string username, string password [, string db])
```

OCIPLogon() creates a persistent connection to an Oracle 8 database and logs on. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

See also **OCILogon()** and **OCINLogon()**.

OCINLogon (PHP 3>= 3.0.8, PHP 4)

Connect to an Oracle database and log on using a new connection. Returns a new session.

```
int OCINLogon (string username, string password [, string db])
```

OCINLogon() creates a new connection to an Oracle 8 database and logs on. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

OCINLogon() forces a new connection. This should be used if you need to isolate a set of transactions. By default, connections are shared at the page level if using **OCILogon()** or at the web server process level if using **OCIPLogon()**. If you have multiple connections open using **OCINLogon()**, all commits and rollbacks apply to the specified connection only.

This example demonstrates how the connections are separated.

Example 1. OCINLogon

```
<?php
print "<HTML><PRE>" ;
$db = "";

$c1 = ocilogon("scott","tiger",$db);
$c2 = ocinlogon("scott","tiger",$db);

function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test
varchar2(64))");
  ociexecute($stmt);
  echo $conn." created table\n\n";
}

function drop_table($conn)
{ $stmt = ociparse($conn,"drop table scott.hallo");
  ociexecute($stmt);
  echo $conn." dropped table\n\n";
}

function insert_data($conn)
{ $stmt = ociparse($conn,"insert into scott.hallo
      values('$conn' || ' ' || to_char(sysdate,'DD-MON-YY HH24:MI:SS'))");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." inserted hallo\n\n";
}

function delete_data($conn)
{ $stmt = ociparse($conn,"delete from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." deleted hallo\n\n";
}

function commit($conn)
{ ocicommit($conn);
  echo $conn." committed\n\n";
}

function rollback($conn)
{ ocirollback($conn);
  echo $conn." rollback\n\n";
}
```

```

}

function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."---selecting\n\n";
  while (ocifetch($stmt))
    echo $conn." <".ociresult($stmt,"TEST").">\n\n";
  echo $conn."---done\n\n";
}

create_table($c1);
insert_data($c1);

select_data($c1);
select_data($c2);

rollback($c1);

select_data($c1);
select_data($c2);

insert_data($c2);
commit($c2);

select_data($c1);

delete_data($c1);
select_data($c1);
select_data($c2);
commit($c1);

select_data($c1);
select_data($c2);

drop_table($c1);
print "</PRE></HTML>";
?>

```

See also **OCILogon()** and **OCIPLogon()**.

OCILogOff (PHP 3>= 3.0.4, PHP 4)

Disconnects from Oracle

```
int OCILogOff (int connection)
```

OCILogOff() closes an Oracle connection.

OCIExecute (PHP 3>= 3.0.4, PHP 4)

Execute a statement

```
int OCIExecute (int statement [, int mode])
```

OCIEexecute() executes a previously parsed statement. (see **OCIParse()**). The optional *mode* allows you to specify the execution-mode (default is OCI_COMMIT_ON_SUCCESS). If you don't want statements to be committed automatically specify OCI_DEFAULT as your mode.

OCICCommit (PHP 3>= 3.0.7, PHP 4)

Commits outstanding transactions

```
int OCICCommit (int connection)
```

OCICCommit() commits all outstanding statements for Oracle connection *connection*.

OCIRollback (PHP 3>= 3.0.7, PHP 4)

Rolls back outstanding transactions

```
int OCIRollback (int connection)
```

OCIRollback() rolls back all outstanding statements for Oracle connection *connection*.

OCINewDescriptor (PHP 3>= 3.0.7, PHP 4)

Initialize a new empty descriptor LOB/FILE (LOB is default)

```
string OCINewDescriptor (int connection [, int type])
```

OCINewDescriptor() Allocates storage to hold descriptors or LOB locators. Valid values for the valid *type* are OCI_D_FILE, OCI_D_LOB, OCI_D_ROWID. For LOB descriptors, the methods load, save, and savefile are associated with the descriptor, for BFILE only the load method exists. See the second example usage hints.

Example 1. OCINewDescriptor

```
<?php
/* This script is designed to be called from a HTML form.
 * It expects $user, $password, $table, $where, and $commitsize
 * to be passed in from the form. The script then deletes
 * the selected rows using the ROWID and commits after each
 * set of $commitsize rows. (Use with care, there is no rollback)
 */
$conn = OCILogon($user, $password);
$stmt = OCIParse($conn,"select rowid from $table $where");
$rowid = OCINewDescriptor($conn,OCI_D_ROWID);
OCIDefineByName($stmt,"ROWID",&$rowid);
OCIEexecute($stmt);
while ( OCIFetch($stmt) ) {
    $nrows = OCIRowCount($stmt);
    $delete = OCIParse($conn,"delete from $table where ROWID = :rid");
    OCIBindByName($delete,:rid,&$rowid,-1,OCI_B_ROWID);
    OCIEexecute($delete);
    print "$nrows\n";
```

```

        if ( ($nrows % $commitsize) == 0 ) {
            OCICCommit($conn);
        }
    }
$nrows = OCIRowCount($stmt);
print "$nrows deleted...\n";
OCIFreeStatement($stmt);
OCILogoff($conn);
?>

<?php
/* This script demonstrates file upload to LOB columns
 * The formfield used for this example looks like this
 * <form action="upload.php3" method="post" enctype="multipart/form-data">
 * <input type="file" name="lob_upload">
 * ...
 */
if(!isset($lob_upload) || $lob_upload == 'none'){
?>
<form action="upload.php3" method="post" enctype="multipart/form-data">
Upload file: <input type="file" name="lob_upload"><br>
<input type="submit" value="Upload"> - <input type="reset">
</form>
<?php
} else {
    // $lob_upload contains the temporary filename of the uploaded file
    $conn = OCILogon($user, $password);
    $lob = OCINewDescriptor($conn, OCI_D_LOB);
    $stmt = OCIParse($conn, "insert into $table (id, the_blob)
                                values(my_seq.NEXTVAL, EMPTY_BLOB()) returning the_blob into :the_blob");
    OCIBindByName($stmt, ':the_blob', &$lob, -1, OCI_B_BLOB);
    OCIExecute($stmt);
    if($lob->savefile($lob_upload)){
        OCICCommit($conn);
        echo "Blob successfully uploaded\n";
    }else{
        echo "Couldn't upload Blob\n";
    }
    OCIFreeDesc($lob);
    OCIFreeStatement($stmt);
    OCILogoff($conn);
}
?>

```

OCIRowCount (PHP 3>= 3.0.7, PHP 4)

Gets the number of affected rows

```
int OCIRowCount (int statement)
```

OCIRowCount() returns the number of rows affected for eg update-statements. This function will not tell you the number of rows that a select will return!

Example 1. OCIRowCount

```
<?php
    print "<HTML><PRE>" ;
```

```

$conn = OCILogon("scott", "tiger");
$stmt = OCIParse($conn, "create table emp2 as select * from emp");
OCIEexecute($stmt);
print OCIRowCount($stmt) . " rows inserted.<BR>";
OCIFreeStatement($stmt);
$stmt = OCIParse($conn, "delete from emp2");
OCIEexecute($stmt);
print OCIRowCount($stmt) . " rows deleted.<BR>";
OCICcommit($conn);
OCIFreeStatement($stmt);
$stmt = OCIParse($conn, "drop table emp2");
OCIEexecute($stmt);
OCIFreeStatement($stmt);
OCILogOff($conn);
print "</PRE></HTML>" ;
?>

```

OCINumCols (PHP 3>= 3.0.4, PHP 4)

Return the number of result columns in a statement

```
int OCINumCols (int stmt)
```

OCINumCols() returns the number of columns in a statement

Example 1. OCINumCols

```

<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn, "select * from emp");
    OCIEexecute($stmt);
    while ( OCIFetch($stmt) ) {
        print "\n";
        $ncols = OCINumCols($stmt);
        for ( $i = 1; $i <= $ncols; $i++ ) {
            $column_name = OCIColumnName($stmt,$i);
            $column_value = OCIResult($stmt,$i);
            print $column_name . ':' . $column_value . "\n";
        }
        print "\n";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>";
    print "</HTML>\n";
?>

```

OCIResult (PHP 3>= 3.0.4, PHP 4)

Returns column value for fetched row

```
mixed OCIResult (int statement, mixed column)
```

OCIResult() returns the data for column *column* in the current row (see **OCIFetch()**). **OCIResult()** will return everything as strings except for abstract types (ROWIDs, LOBs and FILEs).

OCIFetch (PHP 3>= 3.0.4, PHP 4)

Fetches the next row into result-buffer

```
int OCIFetch (int statement)
```

OCIFetch() fetches the next row (for SELECT statements) into the internal result-buffer.

OCIFetchInto (PHP 3>= 3.0.4, PHP 4)

Fetches the next row into result-array

```
int OCIFetchInto (int stmt, array &result [, int mode])
```

OCIFetchInto() fetches the next row (for SELECT statements) into the *result* array. **OCIFetchInto()** will overwrite the previous content of *result*. By default *result* will contain a one-based array of all columns that are not NULL.

The *mode* parameter allows you to change the default behaviour. You can specify more than one flag by simply adding them up (eg OCI_ASSOC+OCI_RETURN_NULLS). The known flags are:

```
OCI_ASSOC Return an associative array.  
OCI_NUM Return an numbered array starting with one. (DEFAULT)  
OCI_RETURN_NULLS Return empty columns.  
OCI_RETURN_LOBS Return the value of a LOB instead of the descriptor.
```

OCIFetchStatement (PHP 3>= 3.0.8, PHP 4)

Fetch all rows of result data into an array.

```
int OCIFetchStatement (int stmt, array &variable)
```

OCIFetchStatement() fetches all the rows from a result into a user-defined array. **OCIFetchStatement()** returns the number of rows fetched.

Example 1. OCIFetchStatement

```
<?php  
/* OCIFetchStatement example mbritton@verinet.com (990624) */  
  
$conn = OCILogon("scott","tiger");  
  
$stmt = OCIParse($conn,"select * from emp");  
  
OCIEexecute($stmt);
```

```

$nrrows = OCIFetchStatement($stmt,$results);
if ( $nrrows > 0 ) {
    print "<TABLE BORDER=\"1\">\n";
    print "<TR>\n";
    while ( list( $key, $val ) = each( $results ) ) {
        print "<TH>$key</TH>\n";
    }
    print "</TR>\n";

    for ( $i = 0; $i < $nrrows; $i++ ) {
        reset($results);
        print "<TR>\n";
        while ( $column = each($results) ) {
            $data = $column['value'];
            print "<TD>$data[$i]</TD>\n";
        }
        print "</TR>\n";
    }
    print "</TABLE>\n";
} else {
    echo "No data found<BR>\n";
}
print "$nrrows Records Selected<BR>\n";

OCIFreeStatement($stmt);
OCILogoff($conn);
?>

```

OCIColumnIsNULL (PHP 3>= 3.0.4, PHP 4)

test whether a result column is NULL

```
int OCIColumnIsNULL (int stmt, mixed column)
```

OCIColumnIsNULL() returns true if the returned column *column* in the result from the statement *stmt* is NULL. You can either use the column-number (1-Based) or the column-name for the *col* parameter.

OCIColumnName (PHP 3>= 3.0.4, PHP 4)

Returns the name of a column.

```
string OCIColumnName (int stmt, int col)
```

OCIColumnName() returns the name of the column corresponding to the column number (1-based) that is passed in.

Example 1. OCIColumnName

```
<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIEexecute($stmt);
```

```

print "<TABLE BORDER=\"1\">" ;
print "<TR>" ;
print "<TH>Name</TH>" ;
print "<TH>Type</TH>" ;
print "<TH>Length</TH>" ;
print "</TR>" ;
$ncols = OCINumCols($stmt) ;
for ( $i = 1; $i <= $ncols; $i++ ) {
    $column_name = OCIColumnName($stmt,$i) ;
    $column_type = OCIColumnType($stmt,$i) ;
    $column_size = OCIColumnSize($stmt,$i) ;
    print "<TR>" ;
    print "<TD>$column_name</TD>" ;
    print "<TD>$column_type</TD>" ;
    print "<TD>$column_size</TD>" ;
    print "</TR>" ;
}
OCIFreeStatement($stmt) ;
OCILogoff($conn) ;
print "</PRE>" ;
print "</HTML>\n" ;
?>

```

See also **OCINumCols()**, **OCIColumnType()**, and **OCIColumnSize()**.

OCIColumnSize (PHP 3>= 3.0.4, PHP 4)

return result column size

```
int OCIColumnSize (int stmt, mixed column)
```

OCIColumnSize() returns the size of the column as given by Oracle. You can either use the column-number (1-Based) or the column-name for the *col* parameter.

Example 1. OCIColumnSize

```
<?php
print "<HTML><PRE>\n" ;
$conn = OCILogon("scott", "tiger");
$stmt = OCIParse($conn,"select * from emp") ;
OCIEexecute($stmt) ;
print "<TABLE BORDER=\"1\">" ;
print "<TR>" ;
print "<TH>Name</TH>" ;
print "<TH>Type</TH>" ;
print "<TH>Length</TH>" ;
print "</TR>" ;
$ncols = OCINumCols($stmt) ;
for ( $i = 1; $i <= $ncols; $i++ ) {
    $column_name = OCIColumnName($stmt,$i) ;
    $column_type = OCIColumnType($stmt,$i) ;
    $column_size = OCIColumnSize($stmt,$i) ;
    print "<TR>" ;
    print "<TD>$column_name</TD>" ;
    print "<TD>$column_type</TD>" ;
    print "<TD>$column_size</TD>" ;
}
```

```

        print "</TR>" ;
    }
    print "</TABLE>" ;
    OCIFreeStatement($stmt) ;
    OCILogoff($conn) ;
    print "</PRE>" ;
    print "</HTML>\n" ;
?>

```

See also **OCINumCols()**, **OCIColumnName()**, and **OCIColumnSize()**.

OCIColumnType (PHP 3>= 3.0.4, PHP 4)

Returns the data type of a column.

```
mixed OCIColumnType (int stmt, int col)
```

OCIColumnType() returns the data type of the column corresponding to the column number (1-based) that is passed in.

Example 1. OCIColumnType

```

<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn, "select * from emp");
    OCIEexecute($stmt);
    print "<TABLE BORDER=\\"1\\">";
    print "<TR>";
    print "<TH>Name</TH>";
    print "<TH>Type</TH>";
    print "<TH>Length</TH>";
    print "</TR>";
    $ncols = OCINumCols($stmt);
    for ( $i = 1; $i <= $ncols; $i++ ) {
        $column_name = OCIColumnName($stmt,$i);
        $column_type = OCIColumnType($stmt,$i);
        $column_size = OCIColumnSize($stmt,$i);
        print "<TR>";
        print "<TD>$column_name</TD>";
        print "<TD>$column_type</TD>";
        print "<TD>$column_size</TD>";
        print "</TR>";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>" ;
    print "</HTML>\n";
?>

```

See also **OCINumCols()**, **OCIColumnName()**, and **OCIColumnSize()**.

OCIServerVersion (PHP 3>= 3.0.4, PHP 4)

Return a string containing server version information.

```
string OCIServerVersion (int conn)
```

Example 1. OCIServerVersion

```
<?php
$conn = OCILogon("scott","tiger");
print "Server Version: " . OCIServerVersion($conn);
OCILogOff($conn);
?>
```

OCIStatementType (PHP 3>= 3.0.5, PHP 4)

Return the type of an OCI statement.

```
string OCIStatementType (int stmt)
```

OCIStatementType() returns one of the following values:

1. "SELECT"
2. "UPDATE"
3. "DELETE"
4. "INSERT"
5. "CREATE"
6. "DROP"
7. "ALTER"
8. "BEGIN"
9. "DECLARE"
10. "UNKNOWN"

Example 1. Code examples

```
<?php
print "<HTML><PRE>";
$conn = OCILogon("scott","tiger");
$sql = "delete from emp where deptno = 10";

$stmt = OCIParse($conn,$sql);
if ( OCIStatementType($stmt) == "DELETE" ) {
    die "You are not allowed to delete from this table<BR>";
}

OCILogoff($conn);
print "</PRE></HTML>";
```

```
?>
```

OCINewCursor (PHP 3>= 3.0.8, PHP 4)

Return a new cursor (Statement-Handle) - use to bind ref-cursors.

```
int OCINewCursor (int conn)
```

OCINewCursor() allocates a new statement handle on the specified connection.

Example 1. Using a REF CURSOR from a stored procedure

```
<?php
// suppose your stored procedure info.output returns a ref cursor in :data

$conn = OCILogon("scott","tiger");
$curs = OCINewCursor($conn);
$stmt = OCIParse($conn,"begin info.output(:data); end;");

ocibindbyname($stmt, "data", &$curs, -1, OCI_B_CURSOR);
ociexecute($stmt);
ociexecute($curs);

while (OCIFetchInto($curs, &$data)) {
    var_dump($data);
}

OCIFreeCursor($stmt);
OCIFreeStatement($curs);
OCILogoff($conn);
?>
```

Example 2. Using a REF CURSOR in a select statement

```
<?php
print "<HTML><BODY>";
$conn = OCILogon("scott","tiger");
$count_cursor = "CURSOR(select count(empno) num_emps from emp ".
               "where emp.deptno = dept.deptno) as EMPCNT from dept";
$stmt = OCIParse($conn, "select deptno,dname,$count_cursor");

ociexecute($stmt);
print "<TABLE BORDER=\\"1\\">";
print "<TR>";
print "<TH>DEPT NAME</TH>";
print "<TH>DEPT #</TH>";
print "<TH># EMPLOYEES</TH>";
print "</TR>";

while (OCIFetchInto($stmt, &$data, OCI_ASSOC)) {
    print "<TR>";
    $dname = $data[ "DNAME" ];
```

```

$deptno = $data[ "DEPTNO" ];
print "<TD>$dname</TD>";
print "<TD>$deptno</TD>";
ociexec($data[ "EMPCNT" ]);
while (OCIFetchInto($data[ "EMPCNT" ],&$subdata,OCI_ASSOC)) {
    $num_emps = $subdata[ "NUM_EMPS" ];
    print "<TD>$num_emps</TD>";
}
print "</TR>";
}
print "</TABLE>";
print "</BODY></HTML>";
OCIFreeStatement($stmt);
OCILogoff($conn);
?>

```

OCIFreeStatement (PHP 3>= 3.0.5, PHP 4)

Free all resources associated with a statement.

```
int OCIFreeStatement (int stmt)
```

OCIFreeStatement() returns true if successful, or false if unsuccessful.

OCIFreeCursor (PHP 3>= 3.0.8, PHP 4)

Free all resources associated with a cursor.

```
int OCIFreeCursor (int stmt)
```

OCIFreeCursor() returns true if successful, or false if unsuccessful.

OCIFreeDesc (PHP 4 >= 4.0b4)

Deletes a large object descriptor.

```
int OCIFreeDesc (object lob)
```

OCIFreeDesc() returns true if successful, or false if unsuccessful.

OCIParse (PHP 3>= 3.0.4, PHP 4)

Parse a query and return a statement

```
int OCIParse (int conn, strint query)
```

OCIParse() parses the *query* using *conn*. It returns the statement identity if the query is valid, false if not. The *query* can be any valid SQL statement.

OCIError (PHP 3>= 3.0.7, PHP 4)

Return the last error of stmt|conn|global. If no error happened returns false.

```
array OCIError ([int stmt/conn/global])
```

OCIError() returns the last error found. If the optional *stmt / conn / global* is not provided, the last error encountered is returned. If no error is found, **OCIError()** returns false. **OCIError()** returns the error as an associative array. In this array, *code* consists the oracle error code and *message* the oracle errorstring.

OCIInternalDebug (PHP 3>= 3.0.4, PHP 4)

Enables or disables internal debug output. By default it is disabled

```
void OCIInternalDebug (int onoff)
```

OCIInternalDebug() enables internal debug output. Set *onoff* to 0 to turn debug output off, 1 to turn it on.

XLVII. OpenSSL functions

This module uses the functions of OpenSSL (<http://www.openssl.org/>) for generation and verification of signatures and for sealing (encrypting) and opening (decrypting) data. You need to use OpenSSL >= 0.9.6 with this module.

OpenSSL offers many features that this module currently doesn't support. Some of these may be added in the future.

openssl_free_key (unknown)

Free key resource

```
void openssl_free_key (int key_identifier)
```

openssl_free_key() frees the key associated with the specified *key_identifier* from memory.

openssl_get_privatekey (unknown)

Prepare a PEM formatted private key for use

```
int openssl_get_privatekey (string key [, string passphrase])
```

Returns a positive key identifier on success, or false on error.

openssl_get_privatekey() parses the PEM formatted private key specified by *key* and prepares it for use by other functions. The optional parameter *passphrase* must be used if the specified key is encrypted (protected by a passphrase).

openssl_get_publickey (unknown)

Extract public key from certificate and prepare it for use

```
int openssl_get_publickey (string certificate)
```

Returns a positive key identifier on success, or false on error.

openssl_get_publickey() extracts the public key from an X.509 certificate specified by *certificate* and prepares it for use by other functions.

openssl_open (unknown)

Open sealed data

```
bool openssl_open (string sealed_data, string open_data, string env_key, int priv_key_id)
```

Returns true on success, or false on error. If successful the opened data is returned in *open_data*.

openssl_open() opens (decrypts) *sealed_data* using the private key associated with the key identifier *priv_key_id* and the envelope key *env_key*. The envelope key is generated when the data are sealed and can only be used by one specific private key. See **openssl_seal()** for more information.

Example 1. openssl_open() example

```
// $sealed and $env_key are assumed to contain the sealed data
// and our envelope key, both given to us by the sealer.

// fetch private key from file and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/key.pem", "r");
```

```

$priv_key = fread($fp, 8192);
fclose($fp);
$pkeyid = openssl_get_privatekey($priv_key);

// decrypt the data and store it in $open
if (openssl_open($sealed, $open, $env_key, $pkeyid))
    echo "here is the opened data: ", $open;
else
    echo "failed to open data";

// free the private key from memory
openssl_free_key($pkeyid);

```

See also **openssl_seal()**.

openssl_seal (unknown)

Seal (encrypt) data

```
int openssl_seal (string data, string sealed_data, array env_keys, array pub_key_ids)
```

Returns the length of the sealed data on success, or false on error. If successful the sealed data is returned in *sealed_data*, and the envelope keys in *env_keys*.

openssl_seal() seals (encrypts) *data* by using RC4 with a randomly generated secret key. The key is encrypted with each of the public keys associated with the identifiers in *pub_key_ids* and each encrypted key is returned in *env_keys*. This means that one can send sealed data to multiple recipients (provided one has obtained their public keys). Each recipient must receive both the sealed data and the envelope key that was encrypted with the recipient's public key.

Example 1. openssl_seal() example

```

// $data is assumed to contain the data to be sealed

// fetch public keys for our recipients, and ready them
$fp = fopen("/src/openssl-0.9.6/demos/maurice/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
$pk1 = openssl_get_publickey($cert);
// Repeat for second recipient
$fp = fopen("/src/openssl-0.9.6/demos/sign/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
$pk2 = openssl_get_publickey($cert);

// seal message, only owners of $pk1 and $pk2 can decrypt $sealed with keys
// $ekeys[0] and $ekeys[1] respectively.
openssl_seal($data, $sealed, $ekeys, array($pk1,$pk2));

// free the keys from memory
openssl_free_key($pk1);
openssl_free_key($pk2);

```

See also **openssl_open()**.

openssl_sign (unknown)

Generate signature

```
bool openssl_sign (string data, string signature, int priv_key_id)
```

Returns true on success, or false on failure. If successful the signature is returned in *signature*.

openssl_sign() computes a signature for the specified *data* by using SHA1 for hashing followed by encryption using the private key associated with *priv_key_id*. Note that the data itself is not encrypted.

Example 1. openssl_sign() example

```
// $data is assumed to contain the data to be signed

// fetch private key from file and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/key.pem", "r");
$priv_key = fread($fp, 8192);
fclose($fp);
$pkeyid = openssl_get_privatekey($priv_key);

// compute signature
openssl_sign($data, $signature, $pkeyid);

// free the key from memory
openssl_free_key($pkeyid);
```

See also **openssl_verify()**.

openssl_verify (unknown)

Verify signature

```
int openssl_verify (string data, string signature, int pub_key_id)
```

Returns 1 if the signature is correct, 0 if it is incorrect, and -1 on error.

openssl_verify() verifies that the *signature* is correct for the specified *data* using the public key associated with *pub_key_id*. This must be the public key corresponding to the private key used for signing.

Example 1. openssl_verify() example

```
// $data and $signature are assumed to contain the data and the signature

// fetch public key from certificate and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
$pubkeyid = openssl_get_publickey($cert);

// state whether signature is okay or not
$ok = openssl_verify($data, $signature, $pubkeyid);
if ($ok == 1)
    echo "good";
```

```
elseif ($ok == 0)
    echo "bad";
else
    echo "ugly, error checking signature";

// free the key from memory
openssl_free_key($pubkeyid);
```

See also **openssl_sign()**.

XLVIII. Oracle functions

Ora_Bind (PHP 3, PHP 4)

bind a PHP variable to an Oracle parameter

```
int ora_bind (int cursor, string PHP variable name, string SQL parameter name, int length
[, int type])
```

Returns true if the bind succeeds, otherwise false. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

This function binds the named PHP variable with a SQL parameter. The SQL parameter must be in the form ":name". With the optional type parameter, you can define whether the SQL parameter is an in/out (0, default), in (1) or out (2) parameter. As of PHP 3.0.1, you can use the constants ORA_BIND_INOUT, ORA_BIND_IN and ORA_BIND_OUT instead of the numbers.

ora_bind must be called after **ora_parse()** and before **ora_exec()**. Input values can be given by assignment to the bound PHP variables, after calling **ora_exec()** the bound PHP variables contain the output values if available.

```
<?php
ora_parse($curs, "declare tmp INTEGER; begin tmp := :in; :out := tmp; :x := 7.77; end;");
ora_bind($curs, "result", ":x", $len, 2);
ora_bind($curs, "input", ":in", 5, 1);
ora_bind($curs, "output", ":out", 5, 2);
$input = 765;
ora_exec($curs);
echo "Result: $result<BR>Out: $output<BR>In: $input";
?>
```

Ora_Close (PHP 3, PHP 4)

close an Oracle cursor

```
int ora_close (int cursor)
```

Returns true if the close succeeds, otherwise false. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

This function closes a data cursor opened with **ora_open()**.

Ora_ColumnName (PHP 3, PHP 4)

get name of Oracle result column

```
string Ora_ColumnName (int cursor, int column)
```

Returns the name of the field/column *column* on the cursor *cursor*. The returned name is in all uppercase letters.

Ora_ColumnSize (PHP 3, PHP 4)

get size of Oracle result column

```
int ora_columnsize (int cursor, int column)
```

Returns the size of the Oracle column *column* on the cursor *cursor*.

Ora_ColumnType (PHP 3, PHP 4)

get type of Oracle result column

```
string ora_columntype (int cursor, int column)
```

Returns the Oracle data type name of the field/column *column* on the cursor *cursor*. The returned type will be one of the following:

```
"VARCHAR2"  
"VARCHAR"  
"CHAR"  
"NUMBER"  
"LONG"  
"LONG RAW"  
"ROWID"  
"DATE"  
"CURSOR"
```

Ora_Commit (PHP 3, PHP 4)

commit an Oracle transaction

```
int ora_commit (int conn)
```

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

This function commits an Oracle transaction. A transaction is defined as all the changes on a given connection since the last commit/rollback, autocommit was turned off or when the connection was established.

Ora_CommitOff (PHP 3, PHP 4)

disable automatic commit

```
int ora_commitoff (int conn)
```

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

This function turns off automatic commit after each **ora_exec()**.

Ora_CommitOff (PHP 3, PHP 4)

enable automatic commit

```
int ora_commiton (int conn)
```

This function turns on automatic commit after each **ora_exec()** on the given connection.

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

Ora_Do (PHP 3, PHP 4)

Parse, Exec, Fetch

```
int ora_do (int conn, string query)
```

This function is quick combination of **ora_parse()**, **ora_exec()** and **ora_fetch()**. It will parse and execute a statement, then fetch the first result row.

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

See also **ora_parse()**, **ora_exec()**, and **ora_fetch()**.

Ora_Error (PHP 3, PHP 4)

get Oracle error message

```
string Ora_Error (int cursor_or_connection)
```

Returns an error message of the form *XXX-NNNNN* where *XXX* is where the error comes from and *NNNNN* identifies the error message.

Note: Support for connection ids was added in 3.0.4.

On UNIX versions of Oracle, you can find details about an error message like this: \$ **oerr ora 00001 00001**,
00000, "unique constraint (%s.%s) violated" // *Cause: An update or insert statement
attempted to insert a duplicate key // For Trusted ORACLE configured in DBMS MAC mode, you
may see // this message if a duplicate entry exists at a different level. // *Action:
Either remove the unique restriction or do not insert the key

Ora_ErrorCode (PHP 3, PHP 4)

get Oracle error code

```
int Ora_ErrorCode (int cursor_or_connection)
```

Returns the numeric error code of the last executed statement on the specified cursor or connection.

* *FIXME: should possible values be listed?*

Note: Support for connection ids was added in 3.0.4.

Ora_Exec (PHP 3, PHP 4)

execute parsed statement on an Oracle cursor

```
int ora_exec (int cursor)
```

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

See also **ora_parse()**, **ora_fetch()**, and **ora_do()**.

Ora_Fetch (PHP 3, PHP 4)

fetch a row of data from a cursor

```
int ora_fetch (int cursor)
```

Returns true (a row was fetched) or false (no more rows, or an error occurred). If an error occurred, details can be retrieved using the **ora_error()** and **ora_errorcode()** functions. If there was no error, **ora_errorcode()** will return 0.

Retrieves a row of data from the specified cursor.

See also **ora_parse()**, **ora_exec()**, and **ora_do()**.

Ora_Fetch_Into (PHP 3, PHP 4)

Fetch a row into the specified result array

```
int ora_fetch_into (int cursor, array result [, int flags])
```

You can fetch a row into an array with this function.

Example 1. Oracle fetch into array

```
<?php
array($results);
ora_fetch_into($cursor, &$results);
```

```
echo $results[0];
echo $results[1];
?>
```

Note that you need to fetch the array by reference.

See also [ora_parse\(\)](#), [ora_exec\(\)](#), [ora_fetch\(\)](#), and [ora_do\(\)](#).

Ora_GetColumn (PHP 3, PHP 4)

get data from a fetched column

```
mixed ora_getcolumn (int cursor, mixed column)
```

Returns the column data. If an error occurs, False is returned and [ora_errorcode\(\)](#) will return a non-zero value. Note, however, that a test for False on the results from this function may be true in cases where there is not error as well (NULL result, empty string, the number 0, the string "0").

Fetches the data for a column or function result.

Ora_Logoff (PHP 3, PHP 4)

close an Oracle connection

```
int ora_logoff (int connection)
```

Returns true on success, false on error. Details about the error can be retrieved using the [ora_error\(\)](#) and [ora_errorcode\(\)](#) functions.

Logs out the user and disconnects from the server.

See also [ora_logon\(\)](#).

Ora_Logon (PHP 3, PHP 4)

open an Oracle connection

```
int ora_logon (string user, string password)
```

Establishes a connection between PHP and an Oracle database with the given username and password.

Connections can be made using SQL*Net by supplying the TNS name to *user* like this:

```
$conn = Ora_Logon("user@TNSNAME", "pass");
```

If you have character data with non-ASCII characters, you should make sure that NLS_LANG is set in your environment. For server modules, you should set it in the server's environment before starting the server.

Returns a connection index on success, or false on failure. Details about the error can be retrieved using the [ora_error\(\)](#) and [ora_errorcode\(\)](#) functions.

Ora_pLogon (PHP 3, PHP 4)

Open a persistent Oracle connection

```
int ora_plogon (string user, string password)
```

Establishes a persistant connection between PHP and an Oracle database with the given username and password.

See also [ora_logon\(\)](#).

Ora_Numcols (PHP 3, PHP 4)

Returns the number of columns

```
int ora_numcols (int cursor_ind)
```

ora_numcols() returns the number of columns in a result. Only returns meaningful values after an parse/exec/fetch sequence.

See also [ora_parse\(\)](#), [ora_exec\(\)](#), [ora_fetch\(\)](#), and [ora_do\(\)](#).

Ora_Numrows (PHP 3, PHP 4)

Returns the number of rows

```
int ora_numrows (int cursor_ind)
```

ora_numrows() returns the number of rows in a result.

Ora_Open (PHP 3, PHP 4)

open an Oracle cursor

```
int ora_open (int connection)
```

Opens an Oracle cursor associated with connection.

Returns a cursor index or False on failure. Details about the error can be retrieved using the [ora_error\(\)](#) and [ora_errorcode\(\)](#) functions.

Ora_Parse (PHP 3, PHP 4)

parse an SQL statement

```
int ora_parse (int cursor_ind, string sql_statement, int defer)
```

This function parses an SQL statement or a PL/SQL block and associates it with the given cursor.

Returns 0 on success or -1 on error.

See also **ora_exec()**, **ora_fetch()**, and **ora_do()**.

Ora_Rollback (PHP 3, PHP 4)

roll back transaction

```
int ora_rollback (int connection)
```

This function undoes an Oracle transaction. (See **ora_commit()** for the definition of a transaction.)

Returns true on success, false on error. Details about the error can be retrieved using the **ora_error()** and **ora_errorcode()** functions.

XLIX. Ovrimos SQL functions

Ovrimos SQL Server, is a client/server, transactional RDBMS combined with Web capabilities and fast transactions.

Ovrimos SQL Server is available at www.ovrimos.com (<http://www.ovrimos.com/>). To enable ovrimos support in PHP just compile php with the '--with-ovrimos' parameter to configure script. You'll need to install the sqlcli library available in the Ovrimos SQL Server distribution.

Example 1. Connect to Ovrimos SQL Server and select from a system table

```
<?php
$conn = ovrimos_connect ("server.domain.com", "8001", "admin", "password");
if ($conn != 0) {
    echo ("Connection ok!");
    $res = ovrimos_exec ($conn, "select table_id, table_name from sys.tables");
    if ($res != 0) {
        echo "Statement ok!";
        ovrimos_result_all ($res);
        ovrimos_free_result ($res);
    }
    ovrimos_close($conn);
}
?>
```

This will just connect to SQL Server.

ovrimos_connect (PHP 4 >= 4.0.3)

Connect to the specified database

```
int ovrimos_connect (string host, string db, string user, string password)
```

ovrimos_connect() is used to connect to the specified database.

ovrimos_connect() returns a connection id (greater than 0) or 0 for failure. The meaning of 'host' and 'port' are those used everywhere in Ovrinos APIs. 'Host' is a host name or IP address and 'db' is either a database name, or a string containing the port number.

Example 1. ovrimos_connect() Example

```
<?php
$conn = ovrimos_connect ("server.domain.com", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res=ovrimos_exec ($conn, "select table_id, table_name from sys.tables");
    if ($res != 0) {
        echo "Statement ok!";
        ovrimos_result_all ($res);
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

The above example will connect to the database and print out the specified table.

ovrimos_close (PHP 4 >= 4.0.3)

Closes the connection to ovrimos

```
void ovrimos_close (int connection)
```

ovrimos_close() is used to close the specified connection.

ovrimos_close() closes a connection to Ovrinos. This has the effect of rolling back uncommitted transactions.

ovrimos_close_all (PHP 4 >= 4.0.3)

Closes all the connections to ovrimos

```
void ovrimos_close_all (void)
```

ovrimos_close_all() is used to close all the connections.

ovrimos_close_all() closes all connections to Ovrinos. This has the effect of rolling back uncommitted transactions.

ovrimos_longreadlen (PHP 4 >= 4.0.3)

Specifies how many bytes are to be retrieved from long datatypes

```
int ovrimos_longreadlen (int result_id, int length)
```

ovrimos_longreadlen() is used to specify how many bytes are to be retrieved from long datatypes.

ovrimos_longreadlen() specifies how many bytes are to be retrieved from long datatypes (long varchar and long varbinary). Default is zero. Regardless of its taking a result_id as an argument, it currently sets this parameter for all result sets. Returns true.

ovrimos_prepare (PHP 4 >= 4.0.3)

Prepares an SQL statement

```
int ovrimos_prepare (int connection_id, string query)
```

ovrimos_prepare() is used to prepare an SQL statement.

ovrimos_prepare() prepares an SQL statement and returns a result_id (or false on failure).

Example 1. Connect to Ovrinos SQL Server and prepare a statement

```
<?php
$conn=ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn!=0) {
    echo "Connection ok!";
    $res=ovrimos_prepare ($conn, "select table_id, table_name
                                from sys.tables where table_id=1");
    if ($res != 0) {
        echo "Prepare ok!";
        if (ovrimos_execute ($res)) {
            echo "Execute ok!\n";
            ovrimos_result_all ($res);
        } else {
            echo "Execute not ok!";
        }
        ovrimos_free_result ($res);
    } else {
        echo "Prepare not ok!\n";
    }
    ovrimos_close ($conn);
}
?>
```

This will connect to Ovrinos SQL Server, prepare a statement and the execute it.

ovrimos_execute (PHP 4 >= 4.0.3)

Executes a prepared SQL statement

```
int ovrimos_execute (int result_id [, array parameters_array])
```

ovrimos_execute() is used to execute an SQL statement.

ovrimos_execute() executes a prepared statement. Returns true or false. If the prepared statement contained parameters (question marks in the statement), the correct number of parameters should be passed in an array. Notice that I don't follow the PHP convention of placing just the name of the optional parameter inside square brackets. I couldn't bring myself on liking it.

ovrimos_cursor (PHP 4 >= 4.0.3)

Returns the name of the cursor

```
int ovrimos_cursor (int result_id)
```

ovrimos_cursor() is used to get the name of the cursor.

ovrimos_cursor() returns the name of the cursor. Useful when wishing to perform positioned updates or deletes.

ovrimos_exec (PHP 4 >= 4.0.3)

Executes an SQL statement

```
int ovrimos_exec (int connection_id, string query)
```

ovrimos_exec() is used to execute an SQL statement.

ovrimos_exec() executes an SQL statement (query or update) and returns a result_id or false. Evidently, the SQL statement should not contain parameters.

ovrimos_fetch_into (PHP 4 >= 4.0.3)

Fetches a row from the result set

```
int ovrimos_fetch_into (int result_id, array result_array [, string how [, int rownumber]])
```

ovrimos_fetch_into() is used to fetch a row from the result set.

ovrimos_fetch_into() fetches a row from the result set into 'result_array', which should be passed by reference. Which row is fetched is determined by the two last parameters. 'how' is one of 'Next' (default), 'Prev', 'First', 'Last', 'Absolute', corresponding to forward direction from current position, backward direction from current position, forward direction from the start, backward direction from the end and absolute position from the start (essentially equivalent to 'first' but needs 'rownumber'). Case is not significant. 'Rownumber' is optional except for absolute positioning. Returns true or false.

Example 1. A fetch into example

```
<?php
$conn=ovrimos_connect ("neptune", "8001", "admin", "password");
if ($conn!=0) {
    echo "Connection ok!";
    $res=ovrimos_exec ($conn,"select table_id, table_name from sys.tables");
```

```

if ($res != 0) {
    echo "Statement ok!";
    if (ovrimos_fetch_into ($res, &$row)) {
        list ($table_id, $table_name) = $row;
        echo "table_id=".$table_id.", table_name=".$table_name."\n";
        if (ovrimos_fetch_into ($res, &$row)) {
            list ($table_id, $table_name) = $row;
            echo "table_id=".$table_id.", table_name=".$table_name."\n";
        } else {
            echo "Next: error\n";
        }
    } else {
        echo "First: error\n";
    }
    ovrimos_free_result ($res);
}
ovrimos_close ($conn);
}
?>

```

This example will fetch a row.

ovrimos_fetch_row (PHP 4 >= 4.0.3)

Fetches a row from the result set

```
int ovrimos_fetch_row (int result_id [, int how [, int row_number]])
```

ovrimos_fetch_row() is used to fetch a row from the result set.

ovrimos_fetch_row() fetches a row from the result set. Column values should be retrieved with other calls. Returns true or false.

Example 1. A fetch row example

```
<?php
$conn = ovrimos_connect ("remote.host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res=ovrimos_exec ($conn, "select table_id, table_name from sys.tables");
    if ($res != 0) {
        echo "Statement ok!";
        if (ovrimos_fetch_row ($res, "First")) {
            $table_id = ovrimos_result ($res, 1);
            $table_name = ovrimos_result ($res, 2);
            echo "table_id=".$table_id.", table_name=".$table_name."\n";
            if (ovrimos_fetch_row ($res, "Next")) {
                $table_id = ovrimos_result ($res, "table_id");
                $table_name = ovrimos_result ($res, "table_name");
                echo "table_id=".$table_id.", table_name=".$table_name."\n";
            } else {
                echo "Next: error\n";
            }
        } else {
            echo "First: error\n";
        }
        ovrimos_free_result ($res);
    }
}
```

```

        ovrimos_close ($conn);
}
?>

```

This will fetch a row and print the result.

ovrimos_result (PHP 4 >= 4.0.3)

Retrieves the output column

```
int ovrimos_result (int result_id, mixed field)
```

ovrimos_result() is used to retrieve the output column.

ovrimos_result() retrieves the output column specified by 'field', either as a string or as an 1-based index.

ovrimos_result_all (PHP 4 >= 4.0.3)

Prints the whole result set as an HTML table

```
int ovrimos_result_all (int result_id [, string format])
```

ovrimos_result_all() is used to print an HTML table containing the whole result set.

ovrimos_result_all() prints the whole result set as an HTML table. Returns true or false.

Example 1. Prepare a statement, execute, and view the result

```

<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_prepare ($conn, "select table_id, table_name
                                from sys.tables where table_id = 7");
    if ($res != 0) {
        echo "Prepare ok!";
        if (ovrimos_execute ($res, array(3))) {
            echo "Execute ok!\n";
            ovrimos_result_all ($res);
        } else {
            echo "Execute not ok!";
        }
        ovrimos_free_result ($res);
    } else {
        echo "Prepare not ok!\n";
    }
    ovrimos_close ($conn);
}
?>

```

This will execute an SQL statement and print the result in an HTML table.

Example 2. Ovrimos_result_all with meta-information

```
<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_exec ($conn, "select table_id, table_name
                                from sys.tables where table_id = 1")
    if ($res != 0) {
        echo "Statement ok! cursor=".ovrimos_cursor ($res)."\n";
        $colnb = ovrimos_num_fields ($res);
        echo "Output columns=".$colnb."\n";
        for ($i=1; $i<=$colnb; $i++) {
            $name = ovrimos_field_name ($res, $i);
            $type = ovrimos_field_type ($res, $i);
            $len = ovrimos_field_len ($res, $i);
            echo "Column ".$i." name=".$name." type=".$type." len=".$len."\n";
        }
        ovrimos_result_all ($res);
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

Example 3. ovrimos_result_all example

```
<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_exec ($conn, "update test set i=5");
    if ($res != 0) {
        echo "Statement ok!";
        echo ovrimos_num_rows ($res)." rows affected\n";
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_num_rows (PHP 4 >= 4.0.3)

Returns the number of rows affected by update operations

```
int ovrimos_num_rows (int result_id)
```

ovrimos_num_rows() is used to get the number of rows affected by update operations.

ovrimos_num_rows() returns the number of rows affected by update operations.

ovrimos_num_fields (PHP 4 >= 4.0.3)

Returns the number of columns

```
int ovrimos_num_fields (int result_id)
```

ovrimos_num_fields() is used to get the number of columns.

ovrimos_num_fields() returns the number of columns in a result_id resulting from a query.

ovrimos_field_name (PHP 4 >= 4.0.3)

Returns the output column name

```
int ovrimos_field_name (int result_id, int field_number)
```

ovrimos_field_name() is used to get the output column name.

ovrimos_field_name() returns the output column name at the (1-based) index specified.

ovrimos_field_type (PHP 4 >= 4.0.3)

Returns the (numeric) type of the output column

```
int ovrimos_field_type (int result_id, int field_number)
```

ovrimos_field_type() is used to get the (numeric) type of the output column.

ovrimos_field_type() returns the (numeric) type of the output column at the (1-based) index specified.

ovrimos_field_len (PHP 4 >= 4.0.3)

Returns the length of the output column

```
int ovrimos_field_len (int result_id, int field_number)
```

ovrimos_field_len() is used to get the length of the output column.

ovrimos_field_len() returns the length of the output column at the (1-based) index specified.

ovrimos_field_num (PHP 4 >= 4.0.3)

Returns the (1-based) index of the output column

```
int ovrimos_field_num (int result_id, string field_name)
```

ovrimos_field_num() is used to get the (1-based) index of the output column.

ovrimos_field_num() returns the (1-based) index of the output column specified by name, or false.

ovrimos_free_result (PHP 4 >= 4.0.3)

Frees the specified result_id

```
int ovrimos_free_result (int result_id)
```

ovrimos_free_result() is used to free the result_id.

ovrimos_free_result() frees the specified result_id. Returns true.

ovrimos_commit (PHP 4 >= 4.0.3)

Commits the transaction

```
int ovrimos_commit (int connection_id)
```

ovrimos_commit() is used to commit the transaction.

ovrimos_commit() commits the transaction.

ovrimos_rollback (PHP 4 >= 4.0.3)

Rolls back the transaction

```
int ovrimos_rollback (int connection_id)
```

ovrimos_rollback() is used to roll back the transaction.

ovrimos_rollback() rolls back the transaction.

L. Output Control Functions

The Output Control functions allow you to control when output is sent from the script. This can be useful in several different situations, especially if you need to send headers to the browser after your script has began outputting data. The Output Control functions do not affect headers sent using **header()** or **setcookie()**, only functions such as **echo()** and data between blocks of PHP code.

Example 1. Output Control example

```
<?php  
  
ob_start();  
echo "Hello\n";  
  
setcookie ("cookiename", "cookiedata");  
  
ob_end_flush();  
  
?>
```

In the above example, the output from **echo()** would be stored in the output buffer until **ob_end_flush()** was called. In the mean time, the call to **setcookie()** successfully stored a cookie without causing an error. (You can not normally send headers to the browser after data has already been sent.)

See also **header()** and **setcookie()**.

flush (PHP 3, PHP 4)

Flush the output buffer

```
void flush(void);
```

Flushes the output buffers of PHP and whatever backend PHP is using (CGI, a web server, etc.) This effectively tries to push all the output so far to the user's browser.

Note: **flush()** has no effect on the buffering scheme of your webserver or the browser on the client side.

Several servers, especially on Win32, will still buffer the output from your script until it terminates before transmitting the results to the browser.

Even the browser may buffer its input before displaying it. Netscape, for example, buffers text until it receives an end-of-line or the beginning of a tag, and it won't render tables until the </table> tag of the outermost table is seen.

ob_start (PHP 4)

Turn on output buffering

```
void ob_start ([string output_callback])
```

This function will turn output buffering on. While output buffering is active no output is sent from the script, instead the output is stored in an internal buffer.

The contents of this internal buffer may be copied into a string variable using **ob_get_contents()**. To output what is stored in the internal buffer, use **ob_end_flush()**. Alternatively, **ob_end_clean()** will silently discard the buffer contents.

An optional output_callback function may be specified. This function takes a string as a parameter and returns a string. The function will be called at **ob_end_flush()** time and will receive the contents of the output buffer as its parameter. It must return a new output buffer as a result, which is what will be printed.

Output buffers are stackable, that is, you may call **ob_start()** while another **ob_start()** is active. Just make sure that you call **ob_end_flush()** the appropriate number of times. If multiple output callback functions are active, output is being filtered sequentially through each of them in nesting order.

Example 1. Callback function example

```
<?php
function c($str) {
    // Druu Chunusun mut dum Kuntrubuß...
    return nl2br(ereg_replace("[aeiou]", "u", $str));
}

function d($str) {
    return strip_tags($str);
}
?>

<?php ob_start("c"); ?>
Drei Chinesen mit dem Kontrabaß...
<?php ob_start("d"); ?>
<h1>..saßen auf der Straße und erzählten sich was...</h1>
<?php ob_end_flush(); ?>
... da kam die Polizei, ja was ist denn das?
```

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

See also **ob_get_contents()**, **ob_end_flush()**, **ob_end_clean()**, and **ob_implicit_flush()**

ob_get_contents (PHP 4)

Return the contents of the output buffer

```
string ob_get_contents(void);
```

This will return the contents of the output buffer or FALSE, if output buffering isn't active.

See also **ob_start()** and **ob_get_length()**.

ob_get_length (PHP 4 >= 4.0.2)

Return the length of the output buffer

```
string ob_get_length(void);
```

This will return the length of the contents in the output buffer or FALSE, if output buffering isn't active.

See also **ob_start()** and **ob_get_contents()**.

ob_end_flush (PHP 4)

Flush (send) the output buffer and turn off output buffering

```
void ob_end_flush(void);
```

This function will send the contents of the output buffer (if any) and turn output buffering off. If you want to further process the buffer's contents you have to call **ob_get_contents()** before **ob_end_flush()** as the buffer contents are discarded after **ob_end_flush()** is called.

See also **ob_start()**, **ob_get_contents()**, and **ob_end_clean()**.

ob_end_clean (PHP 4)

Clean (erase) the output buffer and turn off output buffering

```
void ob_end_clean(void);
```

This function discards the contents of the output buffer and turns off output buffering.

See also **ob_start()** and **ob_end_flush()**.

ob_implicit_flush (PHP 4 >= 4.0b4)

Turn implicit flush on/off

```
void ob_implicit_flush ([int flag])
```

ob_implicit_flush() will turn implicit flushing on or off (if no *flag* is given, it defaults to on). Implicit flushing will result in a flush operation after every output call, so that explicit calls to **flush()** will no longer be needed.

Turning implicit flushing on will disable output buffering, the output buffers current output will be sent as if **ob_end_flush()** had been called.

See also **flush()**, **ob_start()**, and **ob_end_flush()**.

LI. PDF functions

Introduction

You can use the PDF functions in PHP to create PDF files if you have the PDF library by Thomas Merz (available at <http://www.pdflib.com/pdflib/index.html>; you will also need the JPEG library (<ftp://ftp.uu.net/graphics/jpeg/>) and the TIFF library (<http://www.libtiff.org/>) to compile this. These two libs also quite often make problems when configuring php. Follow the messages of configure to fix possible problems.

Please consult the excellent documentation for pdflib shipped with the source distribution of pdflib. It provides a very good overview of what pdflib capable of doing. Most of the functions in pdflib and the PHP module have the same name. The parameters are also identical. You should also understand some of the concepts of PDF or Postscript to efficiently use this module. All lengths and coordinates are measured in Postscript points. There are generally 72 PostScript points to an inch, but this depends on the output resolution.

There is another PHP module for pdf document creation based on FastIO's (<http://www.fastio.com/>). ClibPDF. It has a slightly different API. Check the [ClibPDF functions](#) section for details.

The pdf module introduces two new type of variable. It is called *pdfdoc* *pdfdoc* is a pointer to a pdf document and almost all functions need it as its first parameter.

Confusion with old pdflib versions

Since the very begining of PDF support in PHP — starting with pdflib 0.6 — there has been tons of changes especially to the pdflib API. Most of these changes has been somehow covered by PHP, some has even required changes to the PHP API. Since pdflib 3.x the API seems to be stabilized and PHP 4 has adopted the version as a minimum requirement for PDF support. The consequence will be that many functions will disappear or be replaced by alternatives sooner or later. Support for pdflib 0.6 is already completely given up. The following table list all the functions which are deprecated in PHP 4.02 and should be replaced by their new versions.

Table 1. Deprecated functions and its replacements

Old function	Replacement
<code>pdf_put_image()</code>	Not needed anymore.
<code>pdf_get_font()</code>	<code>pdf_get_value()</code> passing "font" as the second parameter.
<code>pdf_get_fontsize()</code>	<code>pdf_get_value()</code> passing "fontsize" as the second parameter.
<code>pdf_get_fontname()</code>	<code>pdf_get_parameter()</code> passing "fontname" as the second parameter.
<code>pdf_set_info_creator()</code>	<code>pdf_set_info()</code> passing "Creator" as the second parameter.
<code>pdf_set_info_title()</code>	<code>pdf_set_info()</code> passing "Title" as the second parameter.
<code>pdf_set_info_subject()</code>	<code>pdf_set_info()</code> passing "Subject" as the second parameter.
<code>pdf_set_info_author()</code>	<code>pdf_set_info()</code> passing "Author" as the second parameter.
<code>pdf_set_info_keywords()</code>	<code>pdf_set_info()</code> passing "Keywords" as the second parameter.
<code>pdf_set_leading()</code>	<code>pdf_set_value()</code> passing "leading" as the second parameter.

Old function	Replacement
<code>pdf_set_text_rendering()</code>	<code>pdf_set_value()</code> passing "textrendering" as the second parameter.
<code>pdf_set_text_rise()</code>	<code>pdf_set_value()</code> passing "textrise" as the second parameter.
<code>pdf_set_horiz_scaling()</code>	<code>pdf_set_value()</code> passing "horizscaling" as the second parameter.
<code>pdf_set_text_matrix()</code>	Not available anymore
<code>pdf_set_char_spacing()</code>	<code>pdf_set_value()</code> passing "charspacing" as the second parameter.
<code>pdf_set_word_spacing()</code>	<code>pdf_set_value()</code> passing "wordspacing" as the second parameter.
<code>pdf_set_transition()</code>	<code>pdf_set_parameter()</code> passing "transition" as the second parameter.
<code>pdf_set_duration()</code>	<code>pdf_set_value()</code> passing "duration" as the second parameter.
<code>pdf_open_gif()</code>	<code>pdf_open_image_file()</code> passing "gif" as the second parameter.
<code>pdf_open_jpeg()</code>	<code>pdf_open_image_file()</code> passing "jpeg" as the second parameter.
<code>pdf_open_tiff()</code>	<code>pdf_open_image_file()</code> passing "tiff" as the second parameter.
<code>pdf_open_png()</code>	<code>pdf_open_image_file()</code> passing "png" as the second parameter.
<code>pdf_get_imagewidth()</code>	<code>pdf_get_value()</code> passing "imagewidth" as the second parameter and the image as the third parameter.
<code>pdf_get_imageheight()</code>	<code>pdf_get_value()</code> passing "imageheight" as the second parameter and the image as the third parameter.
0	0

Hints for installation of pdflib 3.x

Since version 3.0 of pdflib you should configure pdflib with the option `-enable-shared-pdflib`.

Issues with older versions of pdflib

If you use pdflib 2.01 check how the lib was installed. There should be a file or a to link `libpdf.so`. Version 2.01 just creates a lib with the name `libpdf2.01.so` which cannot be found when linking the test programm in `configure`. You will have to create a symbolic link from `libpdf.so` to `libpdf2.01.so`.

Version 2.20 of pdflib has introduced more changes to its API and support for chinese and japanese fonts. This unfortunately causes some changes of the pdf module of php4 (not php3). If you use pdflib 2.20 handle the in memory generation of PDF documents with care. Until pdflib 3.0 is released it might be unstable. The encoding parameter of `pdf_set_font()` has changed to a string. This means that instead of e.g. 4 you have to use 'winansi'.

If you use pdflib 2.30 the `pdf_set_text_matrix()` will have gone. It is not supported any more. In general it is a good advise

to consult the release notes of the used version of pdflib for possible changes.

Any version of PHP 4 after March, 9th 2000 do not support versions of pdflib older than 3.0. PHP 3 on the other hand should not be used with version newer than 2.01.

Examples

Most of the functions are fairly easy to use. The most difficult part is probably to create a very simple pdf document at all. The following example should help to get started. It creates the file `test.pdf` with one page. The page contains the text "Times Roman outlined" in an outlined, 30pt font. The text is also underlined.

Example 1. Creating a PDF document with pdflib

```
<?php
$fp = fopen("test.pdf", "w");
$pdf = pdf_open($fp);
pdf_set_info($pdf, "Author", "Uwe Steinmann");
pdf_set_info($pdf, "Title", "Test for PHP wrapper of PDFlib 2.0");
pdf_set_info($pdf, "Creator", "See Author");
pdf_set_info($pdf, "Subject", "Testing");
pdf_begin_page($pdf, 595, 842);
pdf_add_outline($pdf, "Page 1");
pdf_set_font($pdf, "Times-Roman", 30, "host");
pdf_set_value($pdf, "textrendering", 1);
pdf_show_xy($pdf, "Times Roman outlined", 50, 750);
pdf_moveto($pdf, 50, 740);
pdf_linetoo($pdf, 330, 740);
pdf_stroke($pdf);
pdf_end_page($pdf);
pdf_close($pdf);
fclose($fp);
echo "<A HREF=getpdf.php>finished</A>" ;
?>
```

The script `getpdf.php` just returns the pdf document.

```
<?php
$fp = fopen("test.pdf", "r");
header("Content-type: application/pdf");
fpassthru($fp);
fclose($fp);
?>
```

The pdflib distribution contains a more complex example which creates a series of pages with an analog clock. This example converted into PHP using pdflib looks as the following (you can see the same example in the documentation for the [clipdf module](#)):

Example 2. pdfclock example from pdflib distribution

```
<?php
$pdffilename = "clock.pdf";
$radius = 200;
$margin = 20;
$pagecount = 40;
```

```

$fp = fopen($pdffilename, "w");
$pdf = pdf_open($fp);
pdf_set_info($pdf, "Creator", "pdf_clock.php3");
pdf_set_info($pdf, "Author", "Uwe Steinmann");
pdf_set_info($pdf, "Title", "Analog Clock");

while($pagecount- > 0) {
    pdf_begin_page($pdf, 2 * ($radius + $margin), 2 * ($radius + $margin));

    pdf_set_parameter($pdf, "transition", "wipe");
    pdf_set_value($pdf, "duration", 0.5);

    pdf_translate($pdf, $radius + $margin, $radius + $margin);
    pdf_save($pdf);
    pdf_setrgbcolor($pdf, 0.0, 0.0, 1.0);

    /* minute strokes */
    pdf_setlinewidth($pdf, 2.0);
    for ($alpha = 0; $alpha < 360; $alpha += 6) {
        pdf_rotate($pdf, 6.0);
        pdf_moveto($pdf, $radius, 0.0);
        pdf_lineto($pdf, $radius-$margin/3, 0.0);
        pdf_stroke($pdf);
    }

    pdf_restore($pdf);
    pdf_save($pdf);

    /* 5 minute strokes */
    pdf_setlinewidth($pdf, 3.0);
    for ($alpha = 0; $alpha < 360; $alpha += 30) {
        pdf_rotate($pdf, 30.0);
        pdf_moveto($pdf, $radius, 0.0);
        pdf_lineto($pdf, $radius-$margin, 0.0);
        pdf_stroke($pdf);
    }

    $ltime = getdate();

    /* draw hour hand */
    pdf_save($pdf);
    pdf_rotate($pdf,-(($ltime['minutes']/60.0)+$ltime['hours']-3.0)*30.0);
    pdf_moveto($pdf, -$radius/10, -$radius/20);
    pdf_lineto($pdf, $radius/2, 0.0);
    pdf_lineto($pdf, -$radius/10, $radius/20);
    pdf_closepath($pdf);
    pdf_fill($pdf);
    pdf_restore($pdf);

    /* draw minute hand */
    pdf_save($pdf);
    pdf_rotate($pdf,-(($ltime['seconds']/60.0)+$ltime['minutes']-15.0)*6.0);
    pdf_moveto($pdf, -$radius/10, -$radius/20);
    pdf_lineto($pdf, $radius * 0.8, 0.0);
    pdf_lineto($pdf, -$radius/10, $radius/20);
    pdf_closepath($pdf);
    pdf_fill($pdf);
    pdf_restore($pdf);

    /* draw second hand */
    pdf_setrgbcolor($pdf, 1.0, 0.0, 0.0);
    pdf_setlinewidth($pdf, 2);

```

```
pdf_save($pdf);
pdf_rotate($pdf, -($ltime['seconds'] - 15.0) * 6.0));
pdf_moveto($pdf, -$radius/5, 0.0);
pdf_lineto($pdf, $radius, 0.0);
pdf_stroke($pdf);
pdf_restore($pdf);

/* draw little circle at center */
pdf_circle($pdf, 0, 0, $radius/30);
pdf_fill($pdf);

pdf_restore($pdf);

pdf_end_page($pdf);
}

$pdf = pdf_close($pdf);
fclose($fp);
echo "<A HREF=getpdf.php?filename=". $pdffilename . ">finished</A>";
?>
```

The PHP script `getpdf.php` just outputs the pdf document.

```
<?php
$fp = fopen($filename, "r");
header("Content-type: application/pdf");
fpassthru($fp);
fclose($fp);
?>
```


pdf_set_info (PHP 4 >= 4.0.1)

Fills a field of the document information

```
void pdf_set_info (int pdf_document, stringfieldname, string value)
```

The **pdf_set_info()** function sets an information field of a pdf document. Possible values for the *fieldname* are 'Subject', 'Title', 'Creator', 'Author', 'Keywords' and one user-defined name. It can be called before beginning a page.

Example 1. Setting document information

```
<?php
$fd = fopen("test.pdf", "w");
$pdfdoc = pdf_open($fd);
pdf_set_info($pdfdoc, "Author", "Uwe Steinmann");
pdf_set_info($pdfdoc, "Creator", "Uwe Steinmann");
pdf_set_info($pdfdoc, "Title", "Testing Info Fields");
pdf_set_info($pdfdoc, "Subject", "Test");
pdf_set_info($pdfdoc, "Keywords", "Test, Fields");
pdf_set_info($pdfdoc, "CustomField", "What ever makes sense");
pdf_begin_page($pdfdoc, 595, 842);
pdf_end_page($pdfdoc);
pdf_close($pdfdoc);
?>
```

Note: This function replaces **pdf_set_info_keywords()**, **pdf_set_info_title()**, **pdf_set_info_subject()**, **pdf_set_info_creator()**, **pdf_set_info_sybject()**.

pdf_open (PHP 3>= 3.0.6, PHP 4)

Opens a new pdf document

```
int pdf_open (int file)
```

The **pdf_open()** function opens a new pdf document. The corresponding file has to be opened with **fopen()** and the file descriptor passed as argument *file*. If you do not pass any parameters, the document will be created in memory and outputed page by page either to stdout or to the web browser.

Note: The return value is needed as the first parameter in all other functions writing to the pdf document.

See also **fopen()**, **pdf_close()**.

pdf_close (PHP 3>= 3.0.6, PHP 4)

Closes a pdf document

```
void pdf_close (int pdf_document)
```

The **pdf_close()** function closes the pdf document.

See also **pdf_open()**, **fclose()**.

pdf_begin_page (PHP 3>= 3.0.6, PHP 4)

Starts new page

```
void pdf_begin_page (int pdf document, double width, double height)
```

The **pdf_begin_page()** function starts a new page with height *height* and width *width*. In order to create a valid document you must call this function and **pdf_end_page()** at least once.

See also **pdf_end_page()**.

pdf_end_page (PHP 3>= 3.0.6, PHP 4)

Ends a page

```
void pdf_end_page (int pdf document)
```

The **pdf_end_page()** function ends a page. Once a page is ended it cannot be modified anymore.

See also **pdf_begin_page()**.

pdf_show (PHP 3>= 3.0.6, PHP 4)

Output text at current position

```
void pdf_show (int pdf document, string text)
```

The **pdf_show()** function outputs the string *text* at the current text position using the current font.

See also **pdf_show_xy()**, **pdf_show_boxed()**, **pdf_set_text_pos()**, **pdf_set_font()**.

pdf_show_boxed (PHP 4 >= 4.0RC1)

Output text in a box

```
int pdf_show_boxed (int pdf document, string text, double x-coor, double y-coor, double width, double height, string mode [, string feature])
```

The **pdf_show_boxed()** function outputs the string *text* in a box with its lower left position at (*x-coor*,*y-coor*). The boxes dimension is *height* by *width*. The parameter *mode* determines how the text is type set. If *width* and *height* are zero, the *mode* can be "left", "right" or "center". If *width* or *height* is unequal zero it can also be "justify" and "fulljustify".

If the parameter *feature* is set to "blind", the text is not shown.

Returns the number of characters that could not be processed because they did not fit into the box.

See also **pdf_show()**, **pdf_show_xy()**.

pdf_show_xy (PHP 3>= 3.0.6, PHP 4)

Output text at given position

```
void pdf_show_xy (int pdf_document, string text, double x-coor, double y-coor)
```

The **pdf_show_xy()** function outputs the string *text* at position (*x-coor*, *y-coor*).

See also **pdf_show()**, **pdf_show_boxed()**.

pdf_set_font (PHP 3>= 3.0.6, PHP 4)

Selects a font face and size

```
void pdf_set_font (int pdf_document, string font_name, double size, string encoding [, int embed])
```

The **pdf_set_font()** function sets the current font face, font size and encoding. If you use pdflib 0.6 you will need to provide the Adobe Font Metrics (afm-files) for the font in the font path (default is ./fonts). If you use php3 or a version of pdflib older than 2.20 the fourth parameter *encoding* can take the following values: 0 = builtin, 1 = pdfdoc, 2 = macroman, 3 = macexpert, 4 = winansi. An encoding greater than 4 and less than 0 will default to winansi. winansi is often a good choice. If you use php4 and a version of pdflib >= 2.20 the encoding parameter has changed to a string. Use 'winansi', 'builtin', 'host', 'macroman' etc. instead. If the last parameter is set to 1 the font is embedded into the pdf document otherwise it is not. To embed a font is usually a good idea if the font is not widely spread and you cannot ensure that the person watching your document has access on fonts in the document. A font is only embedded once even if you call **pdf_set_font()** several times.

Note: This function has to be called after **pdf_begin_page()** in order to create a valid pdf document.

Note: If you reference a font in a .upr file make sure the name in the afm file and the font name are the same. Otherwise, the font will be embedded several times (Thanks to Paul Haddon for finding this.)

pdf_set_leading (PHP 3>= 3.0.6, PHP 4)

Sets distance between text lines

```
void pdf_set_leading (int pdf_document, double distance)
```

The **pdf_set_leading()** function sets the distance between text lines. This will be used if text is output by **pdf_continue_text()**.

See also **pdf_continue_text()**.

pdf_set_parameter (PHP 4 >= 4.0RC1)

Sets certain parameters

```
void pdf_set_parameter (int pdf_document, string name, string value)
```

The **pdf_set_parameter()** function sets several parameters of pdflib which are of the type string.

See also **pdf_get_value()**, **pdf_set_value()**, **pdf_get_parameter()**.

pdf_get_parameter (PHP 4 >= 4.0.1)

Gets certain parameters

```
string pdf_get_parameter (int pdf_document, string name [, double modifier])
```

The **pdf_get_parameter()** function gets several parameters of pdflib which are of the type string. The function parameter *modifier* characterizes the parameter to get. If the modifier is not needed it has to be 0 or not passed at all.

See also **pdf_get_value()**, **pdf_set_value()**, **pdf_set_parameter()**.

pdf_set_value (PHP 4 >= 4.0.1)

Sets certain numerical value

```
void pdf_set_value (int pdf_document, string name, double value)
```

The **pdf_set_value()** function sets several numerical parameters of pdflib.

See also **pdf_get_value()**, **pdf_get_parameter()**, **pdf_set_parameter()**.

pdf_get_value (PHP 4 >= 4.0.1)

Gets certain numerical value

```
double pdf_get_value (int pdf_document, string name [, double modifier])
```

The **pdf_get_value()** function gets several numerical parameters of pdflib. The function parameter *modifier* characterizes the parameter to get. If the modifier is not needed it has to be 0 or not passed at all.

See also **pdf_set_value()**, **pdf_get_parameter()**, **pdf_set_parameter()**.

pdf_get_image_height (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Returns height of an image

```
string pdf_get_image_height (int pdf_document, int image)
```

The **pdf_get_image_height()** function returns the heights of a pdf image in pixel.

See also **pdf_open_image_file()**, **pdf_open_memory_image()**, **pdf_get_image_width()**.

pdf_get_image_width (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Returns width of an image

```
string pdf_get_image_width (int pdf document, int image)
```

The **pdf_get_image_width()** function returns the widths of a pdf image in pixel.

See also **pdf_open_image_file()**, **pdf_open_memory_image()**, **pdf_get_image_height()**.

pdf_set_text_rendering (PHP 3>= 3.0.6, PHP 4)

Determines how text is rendered

```
void pdf_set_text_rendering (int pdf document, int mode)
```

The **pdf_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.

pdf_set_horiz_scaling (PHP 3>= 3.0.6, PHP 4)

Sets horizontal scaling of text

```
void pdf_set_horiz_scaling (int pdf document, double scale)
```

The **pdf_set_horiz_scaling()** function sets the horizontal scaling to *scale* percent.

pdf_set_text_rise (PHP 3>= 3.0.6, PHP 4)

Sets the text rise

```
void pdf_set_text_rise (int pdf document, double rise)
```

The **pdf_set_text_rise()** function sets the text rising to *rise* points.

pdf_set_text_matrix (PHP 3>= 3.0.6, PHP 4 <= 4.0b4)

Sets the text matrix

```
void pdf_set_text_matrix (int pdf document, array matrix)
```

The **pdf_set_text_matrix()** function sets a matrix which describes a transformation applied on the current text font. The matrix has to be passed as an array with six elements.

Note: This function is not available anymore since pdflib 2.3

pdf_set_text_pos (PHP 3>= 3.0.6, PHP 4)

Sets text position

```
void pdf_set_text_pos (int pdf document, double x-coor, double y-coor)
```

The **pdf_set_text_pos()** function sets the position of text for the next **pdf_show()** function call.

See also **pdf_show()**, **pdf_show_xy()**.

pdf_set_char_spacing (PHP 3>= 3.0.6, PHP 4)

Sets character spacing

```
void pdf_set_char_spacing (int pdf document, double space)
```

The **pdf_set_char_spacing()** function sets the spacing between characters.

See also **pdf_set_word_spacing()**, **pdf_set_leading()**.

pdf_set_word_spacing (PHP 3>= 3.0.6, PHP 4)

Sets spacing between words

```
void pdf_set_word_spacing (int pdf document, double space)
```

The **pdf_set_word_spacing()** function sets the spacing between words.

See also **pdf_set_char_spacing()**, **pdf_set_leading()**.

pdf_skew (PHP 4 >= 4.0RC1)

Skews the coordinate system

```
void pdf_skew (int pdf document, double alpha, double beta)
```

The **pdf_skew()** function skew the coordinate system by *alpha* (x) and *beta* (y) degrees. *alpha* and *beta* may not be 90 or 270 degrees.

pdf_continue_text (PHP 3>= 3.0.6, PHP 4)

Outputs text in next line

```
void pdf_continue_text (int pdf document, string text)
```

The **pdf_continue_text()** function outputs the string in *text* in the next line. The distance between the lines can be set with **pdf_set_leading()**.

See also **pdf_show_xy()**, **pdf_set_leading()**, **pdf_set_text_pos()**.

pdf_stringwidth (PHP 3>= 3.0.6, PHP 4)

Returns width of text using current font

```
double pdf_stringwidth (int pdf document, string text)
```

The **pdf_stringwidth()** function returns the width of the string in *text* by using the current font. It requires a font to be set before with **pdf_set_font()**.

See also **pdf_set_font()**.

pdf_save (PHP 3>= 3.0.6, PHP 4)

Saves the current environment

```
void pdf_save (int pdf document)
```

The **pdf_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. **pdf_save()** should always be followed by **pdf_restore()** to restore the environment before **pdf_save()**.

See also **pdf_restore()**.

pdf_restore (PHP 3>= 3.0.6, PHP 4)

Restores formerly saved environment

```
void pdf_restore (int pdf document)
```

The **pdf_restore()** function restores the environment saved with **pdf_save()**. It works like the postscript command grestore.

Example 1. Save and Restore

```
<?php pdf_save($pdf);
// do all kinds of rotations, transformations, ...
pdf_restore($pdf) ?>
```

See also **pdf_save()**.

pdf_translate (PHP 3>= 3.0.6, PHP 4)

Sets origin of coordinate system

```
void pdf_translate (int pdf_document, double x-coor, double y-coor)
```

The **pdf_translate()** function sets the origin of coordinate system to the point (*x-coor*, *y-coor*) relativ the current origin. The following example draws a line from (0, 0) to (200, 200) relative to the initial coordinate system. You have to set the current point after **pdf_translate()** and before you start drawing more objects.

Example 1. Translation

```
<?php pdf_moveto($pdf, 0, 0);
pdf_lineto($pdf, 100, 100);
pdf_stroke($pdf);
pdf_translate($pdf, 100, 100);
pdf_moveto($pdf, 0, 0);
pdf_lineto($pdf, 100, 100);
pdf_stroke($pdf);
?>
```

pdf_scale (PHP 3>= 3.0.6, PHP 4)

Sets scaling

```
void pdf_scale (int pdf_document, double x-scale, double y-scale)
```

The **pdf_scale()** function sets the scaling factor in both directions. The following example scales x and y direction by 72. The following line will therefore be drawn one inch in both directions.

Example 1. Scaling

```
<?php pdf_scale($pdf, 72.0, 72.0);
pdf_lineto($pdf, 1, 1);
pdf_stroke($pdf);
?>
```

pdf_rotate (PHP 3>= 3.0.6, PHP 4)

Sets rotation

```
void pdf_rotate (int pdf_document, double angle)
```

The **pdf_rotate()** function sets the rotation in degress to *angle*.

pdf_setflat (PHP 3>= 3.0.6, PHP 4)

Sets flatness

```
void pdf_setflat (int pdf_document, double value)
```

The **pdf_setflat()** function sets the flatness to a value between 0 and 100.

pdf_setlinejoin (PHP 3>= 3.0.6, PHP 4)

Sets linejoin parameter

```
void pdf_setlinejoin (int pdf_document, long value)
```

The **pdf_setlinejoin()** function sets the linejoin parameter between a value of 0 and 2.

pdf_setlinecap (PHP 3>= 3.0.6, PHP 4)

Sets linecap parameter

```
void pdf_setlinecap (int pdf_document, int value)
```

The **pdf_setlinecap()** function sets the linecap parameter between a value of 0 and 2.

pdf_setmiterlimit (PHP 3>= 3.0.6, PHP 4)

Sets miter limit

```
void pdf_setmiterlimit (int pdf_document, double value)
```

The **pdf_setmiterlimit()** function sets the miter limit to a value greater of equal than 1.

pdf_setlinewidth (PHP 3>= 3.0.6, PHP 4)

Sets line width

```
void pdf_setlinewidth (int pdf_document, double width)
```

The **pdf_setlinewidth()** function sets the line width to *width*.

pdf_setdash (PHP 3>= 3.0.6, PHP 4)

Sets dash pattern

```
void pdf_setdash (int pdf_document, double white, double black)
```

The **pdf_setdash()** function sets the dash pattern *white* white points and *black* black points. If both are 0 a solid line is set.

pdf_moveto (PHP 3>= 3.0.6, PHP 4)

Sets current point

```
void pdf_moveto (int pdf document, double x-coor, double y-coor)
```

The **pdf_moveto()** function sets the current point to the coordinates *x-coor* and *y-coor*.

pdf_curveto (PHP 3>= 3.0.6, PHP 4)

Draws a curve

```
void pdf_curveto (int pdf document, double x1, double y1, double x2, double y2, double x3,
double y3)
```

The **pdf_curveto()** function draws a Bezier curve from the current point to the point (*x3*, *y3*) using (*x1*, *y1*) and (*x2*, *y2*) as control points.

See also **pdf_moveto()**, **pdf_lineto()**, **pdf_stroke()**.

pdf_lineto (PHP 3>= 3.0.6, PHP 4)

Draws a line

```
void pdf_lineto (int pdf document, double x-coor, double y-coor)
```

The **pdf_lineto()** function draws a line from the current point to the point with coordinates (*x-coor*, *y-coor*).

See also **pdf_moveto()**, **pdf_curveto()**, **pdf_stroke()**.

pdf_circle (PHP 3>= 3.0.6, PHP 4)

Draws a circle

```
void pdf_circle (int pdf document, double x-coor, double y-coor, double radius)
```

The **pdf_circle()** function draws a circle with center at point (*x-coor*, *y-coor*) and radius *radius*.

See also **pdf_arc()**, **pdf_stroke()**.

pdf_arc (PHP 3>= 3.0.6, PHP 4)

Draws an arc

```
void pdf_arc (int pdf_document, double x-coor, double y-coor, double radius, double start,
double end)
```

The **pdf_arc()** function draws an arc with center at point (*x-coor*, *y-coor*) and radius *radius*, starting at angle *start* and ending at angle *end*.

See also **pdf_circle()**, **pdf_stroke()**.

pdf_rect (PHP 3>= 3.0.6, PHP 4)

Draws a rectangle

```
void pdf_rect (int pdf_document, double x-coor, double y-coor, double width, double
height)
```

The **pdf_rect()** function draws a rectangle with its lower left corner at point (*x-coor*, *y-coor*). This width is set to *width*. This height is set to *height*.

See also **pdf_stroke()**.

pdf_closepath (PHP 3>= 3.0.6, PHP 4)

Closes path

```
void pdf_closepath (int pdf_document)
```

The **pdf_closepath()** function closes the current path. This means, it draws a line from current point to the point where the first line was started. Many functions like **pdf_moveto()**, **pdf_circle()** and **pdf_rect()** start a new path.

pdf_stroke (PHP 3>= 3.0.6, PHP 4)

Draws line along path

```
void pdf_stroke (int pdf_document)
```

The **pdf_stroke()** function draws a line along current path. The current path is the sum of all line drawing. Without this function the line would not be drawn.

See also **pdf_closepath()**, **pdf_closepath_stroke()**.

pdf_closepath_stroke (PHP 3>= 3.0.6, PHP 4)

Closes path and draws line along path

```
void pdf_closepath_stroke (int pdf_document)
```

The **pdf_closepath_stroke()** function is a combination of **pdf_closepath()** and **pdf_stroke()**. It also clears the path.

See also **pdf_closepath()**, **pdf_stroke()**.

pdf_fill (PHP 3>= 3.0.6, PHP 4)

Fills current path

```
void pdf_fill ( int pdf document )
```

The **pdf_fill()** function fills the interior of the current path with the current fill color.

See also **pdf_closepath()**, **pdf_stroke()**, **pdf_setgray_fill()**, **pdf_setgray()**, **pdf_setrgbcolor_fill()**, **pdf_setrgbcolor()**.

pdf_fill_stroke (PHP 3>= 3.0.6, PHP 4)

Fills and strokes current path

```
void pdf_fill_stroke ( int pdf document )
```

The **pdf_fill_stroke()** function fills the interior of the current path with the current fill color and draws current path.

See also **pdf_closepath()**, **pdf_stroke()**, **pdf_fill()**, **pdf_setgray_fill()**, **pdf_setgray()**, **pdf_setrgbcolor_fill()**, **pdf_setrgbcolor()**.

pdf_closepath_fill_stroke (PHP 3>= 3.0.6, PHP 4)

Closes, fills and strokes current path

```
void pdf_closepath_fill_stroke ( int pdf document )
```

The **pdf_closepath_fill_stroke()** function closes, fills the interior of the current path with the current fill color and draws current path.

See also **pdf_closepath()**, **pdf_stroke()**, **pdf_fill()**, **pdf_setgray_fill()**, **pdf_setgray()**, **pdf_setrgbcolor_fill()**, **pdf_setrgbcolor()**.

pdf_endpath (PHP 3>= 3.0.6, PHP 4)

Ends current path

```
void pdf_endpath ( int pdf document )
```

The **pdf_endpath()** function ends the current path but does not close it.

See also **pdf_closepath()**.

pdf_clip (PHP 3>= 3.0.6, PHP 4)

Clips to current path

```
void pdf_clip (int pdf document)
```

The **pdf_clip()** function clips all drawing to the current path.

pdf_setgray_fill (PHP 3>= 3.0.6, PHP 4)

Sets filling color to gray value

```
void pdf_setgray_fill (int pdf document, double gray value)
```

The **pdf_setgray_fill()** function sets the current gray value to fill a path.

See also **pdf_setrgbcolor_fill()**.

pdf_setgray_stroke (PHP 3>= 3.0.6, PHP 4)

Sets drawing color to gray value

```
void pdf_setgray_stroke (int pdf document, double gray value)
```

The **pdf_setgray_stroke()** function sets the current drawing color to the given gray value.

See also **pdf_setrgbcolor_stroke()**.

pdf_setgray (PHP 3>= 3.0.6, PHP 4)

Sets drawing and filling color to gray value

```
void pdf_setgray (int pdf document, double gray value)
```

The **pdf_setgray()** function sets the current drawing and filling color to the given gray value.

See also **pdf_setrgbcolor_stroke()**, **pdf_setrgbcolor_fill()**.

pdf_setrgbcolor_fill (PHP 3>= 3.0.6, PHP 4)

Sets filling color to rgb color value

```
void pdf_setrgbcolor_fill (int pdf document, double red value, double green value, double blue value)
```

The **pdf_setrgbcolor_fill()** function sets the current rgb color value to fill a path.

See also **pdf_setrgbcolor_fill()**.

pdf_setrgbcolor_stroke (PHP 3>= 3.0.6, PHP 4)

Sets drawing color to rgb color value

```
void pdf_setrgbcolor_stroke (int pdf document, double red value, double green value, double blue value)
```

The **pdf_setrgbcolor_stroke()** function sets the current drawing color to the given rgb color value.

See also **pdf_setrgbcolor_stroke()**.

pdf_setrgbcolor (PHP 3>= 3.0.6, PHP 4)

Sets drawing and filling color to rgb color value

```
void pdf_setrgbcolor (int pdf document, double red value, double green value, double blue value)
```

The **pdf_setrgbcolor_stroke()** function sets the current drawing and filling color to the given rgb color value.

See also **pdf_setrgbcolor_stroke()**, **pdf_setrgbcolor_fill()**.

pdf_add_outline (PHP 3>= 3.0.6, PHP 4)

Adds bookmark for current page

```
int pdf_add_outline (int pdf document, string text [, int parent [, int open]])
```

The **pdf_add_outline()** function adds a bookmark with text *text* that points to the current page. The bookmark is inserted as a child of *parent* and is by default open if *open* is not 0. The return value is an identifier for the bookmark which can be used as a parent for other bookmarks. Therefore you can build up hierarchies of bookmarks.

Unfortunately pdflib does not make a copy of the string, which forces PHP to allocate the memory. Currently this piece of memory is not been freed by any PDF function but it will be taken care of by the PHP memory manager.

pdf_set_transition (PHP 3>= 3.0.6, PHP 4)

Sets transition between pages

```
void pdf_set_transition (int pdf document, int transition)
```

The **pdf_set_transition()** 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)

See also **pdf_set_duration()**.

pdf_set_duration (PHP 3>= 3.0.6, PHP 4)

Sets duration between pages

```
void pdf_set_duration (int pdf document, double duration)
```

The **pdf_set_duration()** function set the duration between following pages in seconds.

See also **pdf_set_transition()**.

pdf_open_gif (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Opens a GIF image

```
int pdf_open_gif (int pdf document, string filename)
```

The **pdf_open_gif()** function opens an image stored in the file with the name *filename*. The format of the image has to be gif. The function returns a pdf image identifier.

Note: This function shouldn't be used anymore. Please use the function **pdf_open_image_file()** instead.

Example 1. Including a gif image

```
<?php
$im = pdf_open_gif($pdf, "test.gif");
pdf_place_image($pdf, $im, 100, 100, 1);
pdf_close_image($pdf, $im);
?>
```

See also **pdf_close_image()**, **pdf_open_jpeg()**, **pdf_open_memory_image()**, **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_open_png (PHP 4 >= 4.0RC2)

Opens a PNG image

```
int pdf_open_png (int pdf, string png_file)
```

The **pdf_open_png()** function opens an image stored in the file with the name *filename*. The format of the image has to be png. The function returns a pdf image identifier.

Note: This function shouldn't be used anymore. Please use the function **pdf_open_image_file()** instead.

Example 1. Including a PNG image

```
<?php
$im = pdf_open_png ($pdf, "test.png");
pdf_place_image ($pdf, $im, 100, 100, 1);
pdf_close_image ($pdf, $im);
?>
```

See also **pdf_close_image()**, **pdf_open_jpeg()**, **pdf_open_gif()**, **pdf_open_memory_image()**, **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_open_image_file (PHP 4 >= 4.0RC2)

Reads an image from a file

```
int pdf_open_image_file (int PDF-document, string format, string filename)
```

The function **pdf_open_image_file()** reads an image of format *format* from the file *filename*. Possible formats are 'png', 'tiff', 'jpeg' and 'gif'. The function returns a pdf image identifier.

Example 1. Inserting an image

```
<?php
$pim = pdf_open_image_file($pdf, "png", "picture.png");
pdf_place_image($pdf, $pim, 100, 100, 1);
pdf_close_image($pdf, $pim);
?>
```

See also **pdf_close_image()**, **pdf_open_jpeg()**, **pdf_open_gif()**, **pdf_open_tiff()**, **pdf_open_png()**, **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_open_memory_image (PHP 3>= 3.0.10, PHP 4 >= 4.0b2)

Opens an image created with PHP's image functions

```
int pdf_open_memory_image (int pdf document, int image)
```

The **pdf_open_memory_image()** function takes an image created with the PHP's image functions and makes it available for the pdf document. The function returns a pdf image identifier.

Example 1. Including a memory image

```
<?php
$im = ImageCreate(100, 100);
$col = ImageColorAllocate($im, 80, 45, 190);
ImageFill($im, 10, 10, $col);
$pim = pdf_open_memory_image($pdf, $im);
```

```
ImageDestroy($im);
pdf_place_image($pdf, $pim, 100, 100, 1);
pdf_close_image($pdf, $pim);
?>
```

See also **pdf_close_image()**, **pdf_open_jpeg()**, **pdf_open_gif()**, **pdf_open_png()** **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_open_jpeg (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Opens a JPEG image

```
int pdf_open_jpeg (int pdf document, string filename)
```

The **pdf_open_jpeg()** function opens an image stored in the file with the name *filename*. The format of the image has to be jpeg. The function returns a pdf image identifier.

Note: This function shouldn't be used anymore. Please use the function **pdf_open_image_file()** instead.

See also **pdf_close_image()**, **pdf_open_gif()**, **pdf_open_png()**, **pdf_open_memory_image()**, **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_open_tiff (PHP 4 >= 4.0b4)

Opens a TIFF image

```
int pdf_open_tiff (int PDF-document, string filename)
```

The **pdf_open_tiff()** function opens an image stored in the file with the name *filename*. The format of the image has to be tiff. The function returns a pdf image identifier.

Note: This function shouldn't be used anymore. Please use the function **pdf_open_image_file()** instead.

See also **pdf_close_image()**, **pdf_open_gif()**, **pdf_open_jpeg()**, **pdf_open_png()**, **pdf_open_memory_image()**, **pdf_execute_image()**, **pdf_place_image()**, **pdf_put_image()**.

pdf_close_image (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Closes an image

```
void pdf_close_image (int image)
```

The **pdf_close_image()** function closes an image which has been opened with any of the **pdf_open_xxx()** functions.

See also **pdf_open_jpeg()**, **pdf_open_gif()**, **pdf_open_memory_image()**.

pdf_place_image (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Places an image on the page

```
void pdf_place_image (int pdf document, int image, double x-coor, double y-coor, double scale)
```

The **pdf_place_image()** function places an image on the page at position (*x-coor*, *x-coor*). The image can be scaled at the same time.

See also **pdf_put_image()**.

pdf_put_image (PHP 3>= 3.0.7, 4.0b2 - 4.0b4 only)

Stores an image in the PDF for later use

```
void pdf_put_image (int pdf document, int image)
```

The **pdf_put_image()** function places an image in the PDF file without showing it. The stored image can be displayed with the **pdf_execute_image()** function as many times as needed. This is useful when using the same image multiple times in order to keep the file size small. Using **pdf_put_image()** and **pdf_execute_image()** is highly recommended for larger images (several kb) if they show up more than once in the document.

Note: This function has become meaningless with version 2.01 of pdflib. It will just output a warning.

See also **pdf_put_image()**, **pdf_place_image()**, **pdf_execute_image()**.

pdf_execute_image (PHP 3>= 3.0.7, 4.0b2 - 4.0b4 only)

Places a stored image on the page

```
void pdf_execute_image (int pdf document, int image, double x-coor, double y-coor, double scale)
```

The **pdf_execute_image()** function displays an image that has been put in the PDF file with the **pdf_put_image()** function on the current page at the given coordinates.

The image can be scaled while displaying it. A scale of 1.0 will show the image in the original size.

Note: This function has become meaningless with version 2.01 of pdflib. It will just output a warning.

Example 1. Multiple show of an image

```
<?php
$im = ImageCreate(100, 100);
$col1 = ImageColorAllocate($im, 80, 45, 190);
```

```

ImageFill($im, 10, 10, $col1);
$pim = pdf_open_memory_image($pdf, $im);
pdf_put_image($pdf, $pim);
pdf_execute_image($pdf, $pim, 100, 100, 1);
pdf_execute_image($pdf, $pim, 200, 200, 2);
pdf_close_image($pdf, $pim);
?>

```

pdf_add_annotation (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Adds annotation

```
void pdf_add_annotation (int pdf document, double llx, double lly, double urx, double ury,
string title, string content)
```

The **pdf_add_annotation()** adds a note with the lower left corner at (*llx*, *lly*) and the upper right corner at (*urx*, *ury*).

pdf_set_border_style (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Sets style of border around links and annotations

```
void pdf_set_border_style (int pdf document, string style, double width)
```

The **pdf_set_border_style()** function sets the style and width of the surrounding box of links and annotations. The parameter *style* can be 'solid' or 'dashed'.

See also **pdf_set_border_color()**, **pdf_set_border_dash()**.

pdf_set_border_color (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Sets color of border around links and annotations

```
void pdf_set_border_color (int pdf document, double red, double green, double blue)
```

The **pdf_set_border_color()** function sets the color of the surrounding box of links and annotations. The three color components have to have a value between 0.0 and 1.0.

See also **pdf_set_border_style()**, **pdf_set_border_dash()**.

pdf_set_border_dash (PHP 4 >= 4.0.1)

Sets dash style of border around links and annotations

```
void pdf_set_border_dash (int pdf document, double black, double white)
```

The **pdf_set_border_dash()** function sets the lenght of black and white areas of a dashed line of the suroundig box of links and annotations.

See also **pdf_set_border_style()**, **pdf_set_border_color()**.

LII. Verisign Payflow Pro functions

This extension allows you to process credit cards and other financial transactions using Verisign Payment Services, formerly known as Signio (<http://www.verisign.com/payment/>).

These functions are only available if PHP has been compiled with the `-with-pfpro[=DIR]` option. You will require the appropriate SDK for your platform, which may be downloaded from within the manager interface (https://testmanager.signio.com/Downloads/Downloads_secure.htm) once you have registered. If you are going to use this extension in an SSL-enabled webserver or with other SSL components (such as the CURL+SSL extension) you MUST get the beta SDK.

Once you have downloaded the SDK you should copy the files from the `lib` directory of the distribution. Copy the header file `pfpro.h` to `/usr/local/include` and the library file `libpfpro.so` to `/usr/local/lib`.

When using these functions, you may omit calls to `pfpro_init()` and `pfpro_cleanup()` as this extension will do so automatically if required. However the functions are still available in case you are processing a number of transactions and require fine control over the library. You may perform any number of transactions using `pfpro_process()` between the two.

These functions have been added in PHP 4.0.2.

Note: These functions only provide a link to Verisign Payment Services. Be sure to read the Payflow Pro Developers Guide for full details of the required parameters.

pfprom_init (PHP 4 >= 4.0.2)

Initialises the Payflow Pro library

```
void pfprom_init(void);
```

pfprom_init() is used to initialise the Payflow Pro library. You may omit this call, in which case this extension will automatically call **pfprom_init()** before the first transaction.

See also **pfprom_cleanup()**.

pfprom_cleanup (PHP 4 >= 4.0.2)

Shuts down the Payflow Pro library

```
void pfprom_cleanup(void);
```

pfprom_cleanup() is used to shutdown the Payflow Pro library cleanly. It should be called after you have processed any transactions and before the end of your script. However you may omit this call, in which case this extension will automatically call **pfprom_cleanup()** after your script terminates.

See also **pfprom_init()**.

pfprom_process (PHP 4 >= 4.0.2)

Process a transaction with Payflow Pro

```
array pfprom_process (array parameters [, string address [, int port [, int timeout [, string proxy address [, int proxy port [, string proxy logon [, string proxy password]]]]]]])
```

Returns: An associative array containing the response

pfprom_process() processes a transaction with Payflow Pro. The first parameter is an associative array containing keys and values that will be encoded and passed to the processor.

The second parameter is optional and specifies the host to connect to. By default this is "test.signio.com", so you will certainly want to change this to "connect.signio.com" in order to process live transactions.

The third parameter specifies the port to connect on. It defaults to 443, the standard SSL port.

The fourth parameter specifies the timeout to be used, in seconds. This defaults to 30 seconds. Note that this timeout appears to only begin once a link to the processor has been established and so your script could potentially continue for a very long time in the event of DNS or network problems.

The fifth parameter, if required, specifies the hostname of your SSL proxy. The sixth parameter specifies the port to use.

The seventh and eighth parameters specify the logon identity and password to use on the proxy.

The function returns an associative array of the keys and values in the response.

Note: Be sure to read the Payflow Pro Developers Guide for full details of the required parameters.

Example 1. Payflow Pro example

```
<?php

pfpromo_init();

$transaction = array(USER => 'mylogin',
    PWD => 'mypassword',
    TRXTYPE => 'S',
    TENDER => 'C',
    AMT => 1.50,
    ACCT => '4111111111111111',
    EXPDATE => '0904'
);

$response = pfpromo_process($transaction);

if (!$response) {
    die("Couldn't establish link to Verisign.\n");
}

echo "Verisign response code was ".$response[RESULT];
echo ", which means: ".$response[RESPMSG]."\n";

echo "\nThe transaction request: ";
print_r($transaction);

echo "\nThe response: ";
print_r($response);

pfpromo_cleanup();

?>
```

pfpromo_process_raw (PHP 4 >= 4.0.2)

Process a raw transaction with Payflow Pro

```
string pfpromo_process_raw (string parameters [, string address [, int port [, int timeout [, string proxy address [, int proxy port [, string proxy logon [, string proxy password]]]]]]])
```

Returns: A string containing the response.

pfpromo_process_raw() processes a raw transaction string with Payflow Pro. You should really use **pfpromo_process()** instead, as the encoding rules of these transactions are non-standard.

The first parameter in this case is a string containing the raw transaction request. All other parameters are the same as with **pfpromo_process()**. The return value is a string containing the raw response.

Note: Be sure to read the Payflow Pro Developers Guide for full details of the required parameters and encoding rules. You would be well advised to use **pfpromo_process()** instead.

Example 1. Payflow Pro raw example

```
<?php
```

```
pffpro_init();

$response = pffpro_process( "USER=mylogin&PWD[ 5 ]=m&ndy&TRXTYPE=S&TENDER=C&AMT=1.50&ACCT=4111111111111111

if (!$response) {
    die("Couldn't establish link to Verisign.\n");
}

echo "Verisign raw response was ".$response;

pffpro_cleanup();

?>
```

pffpro_version (PHP 4 >= 4.0.2)

Returns the version of the Payflow Pro software

```
string pffpro_version(void);
```

pffpro_version() returns the version string of the Payflow Pro library. At the time of writing, this was L211.

LIII. PHP options & information

assert (PHP 4 >= 4.0b4)

Checks if assertion is false

```
int assert (string|bool assertion)
```

assert() will check the given *assertion* and take appropriate action if its result is *false*.

If the *assertion* is given as a string it will be evaluated as PHP code by **assert()**. The advantages of a string *assertion* are less overhead when assertion checking is off and messages containing the *assertion* expression when an assertion fails.

Assertion should be used as a debugging feature only. You may use them for sanity-checks that test for conditions that should always be true and that indicate some programming errors if not or to check for the presence of certain features like extension functions or certain system limits and features.

Assertions should not be used for normal runtime operations like input parameter checks. As a rule of thumb your code should always be able to work correct if assertion checking is not activated.

The behavior of **assert()** may be configured by **assert_options()** or by .ini-settings described in that functions manual page.

assert_options (PHP 4 >= 4.0b4)

Set/get the various assert flags

```
mixed assert_options (int what [, mixed value])
```

Using **assert_options()** you may set the various **assert()** control options or just query their current settings.

Table 1. assert options

option	ini-parameter	default	description
ASSERT_ACTIVE	assert.active	1	enable assert() evaluation
ASSERT_WARNING	assert.warning	1	issue a PHP warning for each failed assertion
ASSERT_BAIL	assert.bail	0	terminate execution on failed assertions
ASSERT QUIET_EVAL	assert.quiet_eval	0	disable error_reporting during assertion expression evaluation
ASSERT_CALLBACK	assert_callback	(null)	user function to call on failed assertions

assert_options() will return the original setting of any option or *false* on errors.

extension_loaded (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

find out whether an extension is loaded

```
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.

dl (PHP 3, PHP 4)

load a PHP extension at runtime

```
int dl (string library)
```

Loads the PHP extension defined in *library*. See also the [extension_dir](#) configuration directive.

getenv (PHP 3, PHP 4)

Get the value of an environment variable

```
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>).

Note: This function does not work in ISAPI mode.

get_cfg_var (PHP 3, PHP 4)

Get the value of a PHP configuration option.

```
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 (PHP 3, PHP 4)

Get the name of the owner of the current PHP script.

```
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 (PHP 3>= 3.0.6, PHP 4)

Get the current active configuration setting of magic quotes gpc.

```
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 (PHP 3>= 3.0.6, PHP 4)

Get the current active configuration setting of magic_quotes_runtime.

```
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 (PHP 3, PHP 4)

Get time of last page modification.

```
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 (PHP 3, PHP 4)

Get the inode of the current script.

```
int getmyinode (void)
```

Returns the current script's inode, or false on error.

See also **getmyuid()**, **get_current_user()**, **getmypid()**, and **getlastmod()**.

Note: This function is not supported on Windows systems.

getmypid (PHP 3, PHP 4)

Get PHP's process ID.

```
int getmypid (void)
```

Returns the current PHP process ID, or false on error.

Warning

Process IDs are not unique, thus they are a weak entropy source. We recommend against relying on pids in security-dependent contexts.

See also **getmyuid()**, **get_current_user()**, **getmyinode()**, and **getlastmod()**.

getmyuid (PHP 3, PHP 4)

Get PHP script owner's UID.

```
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 (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Get the current resource usages.

```
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 on `getrusage(2)` for more details.

ini_alter (PHP 4)

Change the value of a configuration option

```
string ini_alter (string varname, string newvalue)
```

Changes the value of a configuration option, returns `false` on failure, and the previous value of the configuration option on success.

Note: This is an alias of `ini_set()`

See also `ini_get()`, `ini_restore()`, `ini_set()`

ini_get (PHP 4)

Get the value of a configuration option

```
string ini_get (string varname)
```

Returns the value of the configuration option on success, `false` on failure.

See also `ini_alter()`, `ini_restore()`, `ini_set()`

ini_restore (PHP 4)

Restore the value of a configuration option

```
string ini_restore (string varname)
```

Restores a given configuration option to its original value.

See also `ini_alter()`, `ini_get()`, `ini_set()`

ini_set (PHP 4 >= 4.0RC1)

Set the value of a configuration option

```
string ini_set (string varname, string newvalue)
```

Sets the value of the given configuration option. Returns the old value on success, `false` on failure.

See also `ini_alter()`, `ini_get()`, `ini_restore()`

phpcredits (PHP 4)

Prints out the credits for PHP.

```
void phpcredits (int flag)
```

This function prints out the credits listing the PHP developers, modules, etc. It generates the appropriate HTML codes to insert the information in a page. A parameter indicating what will be printed (a pre-defined constant flag, see table below) needs to be passed. For example to print the general credits, you will use somewhere in your code:

```
...
phpcredits(CREDITS_GENERAL);
...
```

And if you want to print the core developers and the documentation group, in a page of its own, you will use:

```
<?php
phpcredits(CREDITS_GROUP + CREDITS_DOCS + CREDITS_FULLPAGE);
?>
```

And if you feel like embedding all the credits in your page, then code like the one below will do it:

```
<html>
<head>
<title>My credits page</title>
</head>
<body>
<?php
// some code of your own
phpcredits(CREDITS_ALL + CREDITS_FULLPAGE);
// some more code
?>
</body>
</html>
```

Table 1. Pre-defined `phpcredits()` flags

name	description
CREDITS_ALL	All the credits, equivalent to using: CREDITS_DOCS + CREDITS_GENERAL + CREDITS_GROUP + CREDITS_MODULES + CREDITS_FULLPAGE. It generates a complete stand-alone HTML page with the appropriate tags.
CREDITS_DOCS	The credits for the documentation team

name	description
CREDITS_FULLSCREEN	Usually used in combination with the other flags. Indicates that the a complete stand-alone HTML page needs to be printed including the information indicated by the other flags.
CREDITS_GENERAL	General credits: Language design and concept, PHP 4.0 authors and SAPI module.
CREDITS_GROUP	A list of the core developers
CREDITS_MODULES	A list of the extension modules for PHP, and their authors
CREDITS_SAPI	A list of the server API modules for PHP, and their authors

See also **phpinfo()**, **phpversion()**, **php_logo_guid()**.

phpinfo (PHP 3, PHP 4)

Output lots of PHP information.

```
int phpinfo ([int what])
```

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 PHP License.

The output may be customized by passing one or more of the following values ored together in the optional parameter *what*.

- INFO_GENERAL
- INFO_CREDITS
- INFO_CONFIGURATION
- INFO_MODULES
- INFO_ENVIRONMENT
- INFO_VARIABLES
- INFO_LICENSE
- INFO_ALL

See also **phpversion()**, **phpcredits()**, **php_logo_guid()**

phpversion (PHP 3, PHP 4)

Get the current PHP version.

```
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()`**, **`phpcredits()`**, **`php_logo_guid()`**

`php_logo_guid` (PHP 4 >= 4.0b4)

Get the logo guid

```
string php_logo_guid (void)
```

Note: This functionality was added in PHP 4 Beta 4.

See also **`phpinfo()`**, **`phpversion()`**, **`phpcredits()`**

`php_sapi_name` (PHP 4 >= 4.0.1)

Returns the type of interface between web server and PHP

```
string php_sapi_name(void);
```

`Php_sapi_name()` returns a lowercase string which describes the type of interface between web server and PHP (Server API, SAPI). In CGI PHP, this string is "cgi", in mod_php for Apache, this string is "apache" and so on.

Example 1. `Php_sapi_name()` Example

```
$sapi_type = php_sapi_name();
if ($sapi_type == "cgi")
    print "You are using CGI PHP\n";
else
    print "You are not using CGI PHP\n";
```

`php_uname` (PHP 4 >= 4.0.2)

Returns information about the operating system PHP was built on

```
string php_uname(void);
```

`php_uname()` returns a string with a description of the operating system PHP is built on.

Example 1. `php_uname()` Example

```
if (substr/php_uname(), 0, 7) == "Windows") {
    die("Sorry, this script doesn't run on Windows.\n");
}
```

`putenv` (PHP 3, PHP 4)

Set the value of an environment variable.

```
void putenv (string setting)
```

Adds *setting* to the server environment.

Example 1. Setting an Environment Variable

```
putenv ("UNIQID=$uniqid");
```

`set_magic_quotes_runtime` (PHP 3>= 3.0.6, PHP 4)

Set the current active configuration setting of `magic_quotes_runtime`.

```
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` (PHP 3, PHP 4)

limit the maximum execution time

```
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.

Note that `set_time_limit()` has no effect when PHP is running in safe mode. There is no workaround other than turning off safe mode or changing the time limit in the [configuration file](#).

zend_logo_guid (PHP 4 >= 4.0b4)

Get the zend guid

```
string zend_logo_guid (void)
```

Note: This functionality was added in PHP 4 Beta 4.

get_loaded_extensions (PHP 4 >= 4.0b4)

Returns an array with the names of all modules compiled and loaded

```
array get_loaded_extensions (void)
```

This function returns the names of all the modules compiled and loaded in the PHP interpreter.

For example the line below

```
print_r (get_loaded_extensions());
```

will print a list like:

```
Array
(
    [0] => xml
    [1] => wddx
    [2] => standard
    [3] => session
    [4] => posix
    [5] => pgsql
    [6] => pcre
    [7] => gd
    [8] => ftp
    [9] => db
    [10] => Calendar
    [11] => bcmath
)
```

See also: [get_extension_funcs\(\)](#).

get_extension_funcs (PHP 4 >= 4.0b4)

Returns an array with the names of the functions of a module

```
array get_extension_funcs (string module_name)
```

This function returns the names of all the functions defined in the module indicated by *module_name*.

For example the lines below

```
print_r (get_extension_funcs ("xml"));
print_r (get_extension_funcs ("gd"));
```

will print a list of the functions in the modules `xml` and `gd` respectively.

See also: [get_loaded_extensions\(\)](#)

get_required_files (PHP 4 >= 4.0RC2)

Returns an array with the names of the files `require_once()`'d in a script

```
array get_required_files (void)
```

This function returns an associative array of the names of all the files that have been loaded into a script using `require_once()`. The indexes of the array are the file names as used in the `require_once()` without the ".php" extension.

The example below

Example 1. Printing the required and included files

```
<?php

require_once ("local.php");
require_once ("../inc/global.php");

for ($i=1; $i<5; $i++)
    include "util".$i.".php";

echo "Required_once files\n";
print_r (get_required_files());

echo "Included_once files\n";
print_r (get_included_files());
?>
```

will generate the following output:

```
Required_once files
Array
(
    [local] => local.php
    [../inc/global] => /full/path/to/inc/global.php
)

Included_once files
Array
(
    [util1] => util1.php
    [util2] => util2.php
    [util3] => util3.php
    [util4] => util4.php
)
```

Note: As of PHP 4.0.1pl2 this function assumes that the `required_once` files end in the extension ".php", other extensions do not work.

See also: **require_once()**, **include_once()**, **get_included_files()**

get_included_files (PHP 4 >= 4.0RC1)

Returns an array with the names of the files `include_once()`'d in a script

```
array get_included_files (void)
```

This function returns an associative array of the names of all the files that have been loaded into a script using **include_once()**. The indexes of the array are the file names as used in the **include_once()** without the ".php" extension.

Note: As of PHP 4.0.1pl2 this function assumes that the `include_once` files end in the extension ".php", other extensions do not work.

See also: **require_once()**, **include_once()**, **get_required_files()**

LIV. POSIX functions

This module contains an interface to those functions defined in the IEEE 1003.1 (POSIX.1) standards document which are not accessible through other means. POSIX.1 for example defined the open(), read(), write() and close() functions, too, which traditionally have been part of PHP 3 for a long time. Some more system specific functions have not been available before, though, and this module tries to remedy this by providing easy access to these functions.

posix_kill (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Send a signal to a process

```
bool posix_kill (int pid, int sig)
```

Send the signal *sig* to the process with the process identifier *pid*. Returns FALSE, if unable to send the signal, TRUE otherwise.

See also the kill(2) manual page of your POSIX system, which contains additional information about negative process identifiers, the special pid 0, the special pid -1, and the signal number 0.

posix_getpid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the current process identifier

```
int posix_getpid (void )
```

Return the process identifier of the current process.

posix_getppid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the parent process identifier

```
int posix_getppid (void )
```

Return the process identifier of the parent process of the current process.

posix_getuid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the real user ID of the current process

```
int posix_getuid (void )
```

Return the numeric real user ID of the current process. See also **posix_getpwuid()** for information on how to convert this into a useable username.

posix_geteuid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the effective user ID of the current process

```
int posix_geteuid (void )
```

Return the numeric effective user ID of the current process. See also **posix_getpwuid()** for information on how to convert this into a useable username.

posix_getgid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the real group ID of the current process

```
int posix_getgid (void )
```

Return the numeric real group ID of the current process. See also **posix_getgrgid()** for information on how to convert this into a useable group name.

posix_getegid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the effective group ID of the current process

```
int posix_getegid (void )
```

Return the numeric effective group ID of the current process. See also **posix_getgrgid()** for information on how to convert this into a useable group name.

posix_setuid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Set the effective UID of the current process

```
bool posix_setuid (int uid)
```

Set the real user ID of the current process. This is a privileged function and you need appropriate privileges (usually root) on your system to be able to perform this function.

Returns TRUE on success, FALSE otherwise. See also **posix_setgid()**.

posix_setgid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Set the effective GID of the current process

```
bool posix_setgid (int gid)
```

Set the real group ID of the current process. This is a privileged function and you need appropriate privileges (usually root) on your system to be able to perform this function. The appropriate order of function calls is **posix_setgid()** first, **posix_setuid()** last.

Returns TRUE on success, FALSE otherwise.

posix_getgroups (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the group set of the current process

```
array posix_getgroups (void )
```

Returns an array of integers containing the numeric group ids of the group set of the current process. See also [posix_getrgid\(\)](#) for information on how to convert this into useable group names.

posix_getlogin (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return login name

```
string posix_getlogin (void )
```

Returns the login name of the user owning the current process. See [posix_getpwnam\(\)](#) for information how to get more information about this user.

posix_getpgrp (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return the current process group identifier

```
int posix_getpgrp (void )
```

Return the process group identifier of the current process. See POSIX.1 and the [getpgrp\(2\)](#) manual page on your POSIX system for more information on process groups.

posix_setsid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Make the current process a session leader

```
int posix_setsid (void )
```

Make the current process a session leader. See POSIX.1 and the [setsid\(2\)](#) manual page on your POSIX system for more informations on process groups and job control. Returns the session id.

posix_setpgid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

set process group id for job control

```
int posix_setpgid (int pid, int pgid)
```

Let the process *pid* join the process group *pgid*. See POSIX.1 and the [setsid\(2\)](#) manual page on your POSIX system for more informations on process groups and job control. Returns TRUE on success, FALSE otherwise.

posix_getpgid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Get process group id for job control

```
int posix_getpgid (int pid)
```

Returns the process group identifier of the process *pid*.

This is not a POSIX function, but is common on BSD and System V systems. If your system does not support this function at system level, this PHP function will always return FALSE.

posix_getsid (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Get the current sid of the process

```
int posix_getsid (int pid)
```

Return the sid of the process *pid*. If *pid* is 0, the sid of the current process is returned.

This is not a POSIX function, but is common on System V systems. If your system does not support this function at system level, this PHP function will always return FALSE.

posix_uname (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Get system name

```
array posix_uname (void )
```

Returns a hash of strings with information about the system. The indices of the hash are

- sysname - operating system name (e.g. Linux)
- nodename - system name (e.g. valiant)
- release - operating system release (e.g. 2.2.10)
- version - operating system version (e.g. #4 Tue Jul 20 17:01:36 MEST 1999)
- machine - system architecture (e.g. i586)
- domainname - DNS domainname (e.g. php.net)

domainname is a GNU extension and not part of POSIX.1, so this field is only available on GNU systems or when using the GNU libc.

Posix requires that you must not make any assumptions about the format of the values, e.g. you cannot rely on three digit version numbers or anything else returned by this function.

posix_times (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Get process times

```
array posix_times (void )
```

Returns a hash of strings with information about the current process CPU usage. The indices of the hash are

- ticks - the number of clock ticks that have elapsed since reboot.
- utime - user time used by the current process.
- stime - system time used by the current process.

- cutime - user time used by current process and children.
- cstime - system time used by current process and children.

posix_ctermid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Get path name of controlling terminal

```
string posix_ctermid (void )
```

Needs to be written.

posix_ttyname (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Determine terminal device name

```
string posix_ttyname (int fd)
```

Needs to be written.

posix_isatty (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Determine if a file descriptor is an interactive terminal

```
bool posix_isatty (int fd)
```

Needs to be written.

posix_getcwd (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Pathname of current directory

```
string posix_getcwd (void )
```

Needs to be written ASAP.

posix_mkfifo (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Create a fifo special file (a named pipe)

```
bool posix_getcwd (string pathname, int mode)
```

Needs to be written ASAP..

posix_getgrnam (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return info about a group by name

```
array posix_getgrnam (string name)
```

Needs to be written.

posix_getgrgid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return info about a group by group id

```
array posix_getgrgid (int gid)
```

Needs to be written.

posix_getpwnam (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return info about a user by username

```
array posix_getpwnam (string username)
```

Returns an associative array containing information about a user referenced by an alphanumeric username, passed in the *username* parameter.

The array elements returned are:

Table 1. The user information array

Element	Description
name	The name element contains the username of the user. This is a short, usually less than 16 character "handle" of the user, not her real, full name. This should be the same as the <i>username</i> parameter used when calling the function, and hence redundant.
passwd	The passwd element contains the user's password in an encrypted format. Often, for example on a system employing "shadow" passwords, an asterisk is returned instead.
uid	User ID of the user in numeric form.
gid	The group ID of the user. Use the function posix_getgrgid() to resolve the group name and a list of its members.
gecos	GECOS is an obsolete term that refers to the finger information field on a Honeywell batch processing system. The field, however, lives on, and its contents have been formalized by POSIX. The field contains a comma separated list containing the user's full name, office phone, office number, and home phone number. On most systems, only the user's full name is available.

Element	Description
dir	This element contains the absolute path to the home directory of the user.
shell	The shell element contains the absolute path to the executable of the user's default shell.

posix_getpwuid (PHP 3>= 3.0.13, PHP 4 >= 4.0b4)

Return info about a user by user id

```
array posix_getpwuid (int uid)
```

Returns an associative array containing information about a user referenced by a numeric user ID, passed in the *uid* parameter.

The array elements returned are:

Table 1. The user information array

Element	Description
name	The name element contains the username of the user. This is a short, usually less than 16 character "handle" of the user, not her real, full name.
passwd	The passwd element contains the user's password in an encrypted format. Often, for example on a system employing "shadow" passwords, an asterisk is returned instead.
uid	User ID, should be the same as the <i>uid</i> parameter used when calling the function, and hence redundant.
gid	The group ID of the user. Use the function posix_getgrgid() to resolve the group name and a list of its members.
gecos	GECOS is an obsolete term that refers to the finger information field on a Honeywell batch processing system. The field, however, lives on, and its contents have been formalized by POSIX. The field contains a comma separated list containing the user's full name, office phone, office number, and home phone number. On most systems, only the user's full name is available.
dir	This element contains the absolute path to the home directory of the user.
shell	The shell element contains the absolute path to the executable of the user's default shell.

posix_getrlimit (PHP 3>= 3.0.10, PHP 4 >= 4.0b4)

Return info about system ressource limits

```
array posix_getrlimit (void )
```

Needs to be written ASAP.

LV. PostgreSQL functions

Postgres, developed originally in the UC Berkeley Computer Science Department, pioneered many of the object-relational concepts now becoming available in some commercial databases. It provides SQL92/SQL3 language support, transaction integrity, and type extensibility. PostgreSQL is an open source descendant of this original Berkeley code.

PostgreSQL is available without cost. The current version is available at www.PostgreSQL.org (<http://www.postgresql.org/>).

Since version 6.3 (03/02/1998) PostgreSQL uses unix domain sockets. A table is shown below describing these new connection possibilities. This socket will be found in `/tmp/.s.PGSQL.5432`. This option can be enabled with the '`-i`' flag to **postmaster** and it's meaning is: "listen on TCP/IP sockets as well as Unix domain sockets".

Table 1. Postmaster and PHP

Postmaster	PHP	Status
postmaster &	<code>pg_connect("dbname=MyDbName");</code>	OK
postmaster -i &	<code>pg_connect("dbname=MyDbName");</code>	OK
postmaster &	<code>pg_connect("host=localhost dbname=MyDbName");</code>	Unable to connect to PostgreSQL server: connectDB() failed: Is the postmaster running and accepting TCP/IP (with -i) connection at 'localhost' on port '5432'? in /path/to/file.php3 on line 20.
postmaster -i &	<code>pg_connect("host=localhost dbname=MyDbName");</code>	OK

One can establish a connection with the following value pairs set in the command string: `$conn = pg_Connect("host=myHost port=myPort tty=myTTY options=myOptions user=myUser password=myPassword dbname=myDB");`

The previous syntax of: `$conn = pg_connect ("host", "port", "options", "tty", "dbname")` has been deprecated.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block. A transaction block starts with a **begin** and if the transaction was valid ends with **commit** or **end**. If the transaction fails the transaction should be closed with **rollback** or **abort**.

Example 1. Using Large Objects

```
<?php
    $database = pg_Connect ("dbname=jacarta");
    pg_exec ($database, "begin");
    $oid = pg_locreate ($database);
    echo ("$oid\n");
    $handle = pg_loopen ($database, $oid, "w");
    echo ("$handle\n");
    pg_lowrite ($handle, "gaga");
    pg_loclose ($handle);
    pg_exec ($database, "commit");
?>
```


pg_close (PHP 3, PHP 4)

Close a PostgreSQL connection

```
bool pg_close (int connection)
```

Returns false if connection is not a valid connection index, true otherwise. Closes down the connection to a PostgreSQL database associated with the given connection index.

Note: This isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

pg_close() will not close persistent links generated by **pg_pconnect()**.

pg_cmtuples (PHP 3, PHP 4)

Returns number of affected tuples

```
int pg_cmtuples (int result_id)
```

Pg_cmtuples() returns the number of tuples (instances) affected by INSERT, UPDATE, and DELETE queries. If no tuple is affected the function will return 0.

Example 1. Pg_cmtuples()

```
<?php
$result = pg_exec ($conn, "INSERT INTO publisher VALUES ('Author')");
$cmtuples = pg_cmtuples ($result);
echo $cmtuples . " <- cmtuples affected.";
?>
```

pg_connect (PHP 3, PHP 4)

Open a PostgreSQL connection

```
int pg_connect (string conn_string)
```

Returns a connection index on success, or false if the connection could not be made. Opens a connection to a PostgreSQL database. The arguments should be within a quoted string.

Example 1. Using pg_connect arguments

```
<?php
$dbconn = pg_Connect ("dbname=mary");
//connect to a database named "mary"
$dbconn2 = pg_Connect ("host=localhost port=5432 dbname=mary");
//connect to a database named "mary" on "localhost" at port "5432"
$dbconn3 = pg_Connect ("user=lamb password=baaaa dbname=mary ");
//connect to a database named "mary" with a username and password
?>
```

The arguments available include *dbname*, *port*, *host*, *tty*, *options*, *user*, and *password*

This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple connections open at once.

The previous syntax of: `$conn = pg_connect ("host", "port", "options", "tty", "dbname")` has been deprecated.

See also [pg_pconnect\(\)](#).

pg_dbname (PHP 3, PHP 4)

Get the database name

```
string pg_dbname (int connection)
```

Returns the name of the database that the given PostgreSQL connection index is connected to, or false if connection is not a valid connection index.

pg_end_copy (PHP 4 >= 4.0.3)

Sync with PostgreSQL backend

```
bool pg_end_copy ([resource connection])
```

pg_end_copy() syncs PostgreSQL frontend with the backend after doing a copy operation. It must be issued or the backend may get "out of sync" with the frontend. Returns TRUE if successfull, FALSE otherwise.

For further details and an example, see also [pg_put_line\(\)](#).

pg_errormessage (PHP 3, PHP 4)

Get the error message string

```
string pg_errormessage (int connection)
```

Returns a string containing the error message, false on failure. Details about the error probably cannot be retrieved using the **pg_errormessage()** function if an error occured on the last database action for which a valid connection exists, this function will return a string containing the error message generated by the backend server.

pg_exec (PHP 3, PHP 4)

Execute a query

```
int pg_exec (int connection, string query)
```

Returns a result index if query could be executed, false on failure or if connection is not a valid connection index. Details about the error can be retrieved using the **pg_ErrorMessage()** function if connection is valid. Sends an SQL statement to the PostgreSQL database specified by the connection index. The connection must be a valid index that

was returned by **pg_Connect()**. The return value of this function is an index to be used to access the results from other PostgreSQL functions.

Note: PHP/FI returned 1 if the query was not expected to return data (inserts or updates, for example) and greater than 1 even on selects that did not return anything. No such assumption can be made in PHP.

pg_fetch_array (PHP 3>= 3.0.1, PHP 4)

Fetch a row as an array

```
array pg_fetch_array (int result, int row [, int result_type])
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

Pg_fetch_array() is an extended version of **pg_fetch_row()**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

The third optional argument *result_type* in **pg_fetch_array()** is a constant and can take the following values: PGSQL_ASSOC, PGSQL_NUM, and PGSQL_BOTH.

Note: *Result_type* was added in PHP 4.0.

An important thing to note is that using **pg_fetch_array()** is NOT significantly slower than using **pg_fetch_row()**, while it provides a significant added value.

For further details, see also **pg_fetch_row()**

Example 1. PostgreSQL fetch array

```
<?php
$conn = pg_pconnect ("dbname=publisher");
if (!$conn) {
    echo "An error occurred.\n";
    exit;
}

$result = pg_Exec ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occurred.\n";
    exit;
}

$arr = pg_fetch_array ($result, 0);
echo $arr[0] . " <- array\n";

$arr = pg_fetch_array ($result, 1);
echo $arr["author"] . " <- array\n";
?>
```

pg_fetch_object (PHP 3>= 3.0.1, PHP 4)

Fetch a row as an object

```
object pg_fetch_object (int result, int row [, int result_type])
```

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

Pg_fetch_object() is similar to **pg_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The third optional argument *result_type* in **pg_fetch_object()** is a constant and can take the following values: PGSQ_L_ASSOC, PGSQ_L_NUM, and PGSQ_L_BOTH.

Note: *Result_type* was added in PHP 4.0.

Speed-wise, the function is identical to **pg_fetch_array()**, and almost as quick as **pg_fetch_row()** (the difference is insignificant).

See also: **pg_fetch_array()** and **pg_fetch_row()**.

Example 1. Postgres fetch object

```
<?php
$database = "verlag";
$db_conn = pg_connect ("host=localhost port=5432 dbname=$database");
if (!$db_conn): ?>
    <H1>Failed connecting to postgres database <?php echo $database ?></H1> <?php
    exit;
endif;

$qu = pg_exec ($db_conn, "SELECT * FROM verlag ORDER BY autor");
$row = 0; // postgres needs a row counter other dbs might not

while ($data = pg_fetch_object ($qu, $row)):
    echo $data->autor." ";
    echo $data->jahr .": ";
    echo $data->titel."<BR>";
    $row++;
endwhile; ?>

<PRE><?php
$fields[] = Array ("autor", "Author");
$fields[] = Array ("jahr", "Year");
$fields[] = Array ("titel", "Title");

$row= 0; // postgres needs a row counter other dbs might not
while ($data = pg_fetch_object ($qu, $row)):
    echo "-----\n";
    reset ($fields);
    while (list (,$item) = each ($fields)):
        echo $item[1].": ".$data->$item[0]."\n";
    endwhile;
    $row++;
endwhile;
echo "-----\n"; ?>
</PRE> <?php
```

```
pg_freeResult ($qu);
pg_close ($db_conn);
?>
```

pg_fetch_row (PHP 3>= 3.0.1, PHP 4)

Get a row as an enumerated array

```
array pg_fetch_row (int result, int row)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

Pg_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.

See also: **pg_fetch_array()**, **pg_fetch_object()**, **pg_result()**.

Example 1. Postgres fetch row

```
<?php
$conn = pg_pconnect ("dbname=publisher");
if (!$conn) {
    echo "An error occurred.\n";
    exit;
}

$result = pg_Exec ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occurred.\n";
    exit;
}

$num = pg_numrows($result);

for ($i=0; $i<$num; $i++) {
    $r = pg_fetch_row($result, $i);

    for ($j=0; $j<count($r); $j++) {
        echo "$r[$j]&nbsp;";
    }

    echo "<BR>";
}
```

?>

pg_fieldisnull (PHP 3, PHP 4)

Test if a field is NULL

```
int pg_fieldisnull (int result_id, int row, mixed field)
```

Test if a field is NULL or not. Returns 0 if the field in the given row is not NULL. Returns 1 if the field in the given row is NULL. Field can be specified as number or fieldname. Row numbering starts at 0.

pg_fieldname (PHP 3, PHP 4)

Returns the name of a field

```
string pg_fieldname (int result_id, int field_number)
```

Pg_fieldname() will return the name of the field occupying the given column number in the given PostgreSQL result identifier. Field numbering starts from 0.

pg_fieldnum (PHP 3, PHP 4)

Returns the field number of the named field

```
int pg_fieldnum (int result_id, string field_name)
```

Pg_fieldnum() will return the number of the column slot that corresponds to the named field in the given PosgreSQL result identifier. Field numbering starts at 0. This function will return -1 on error.

pg_fieldptlen (PHP 3, PHP 4)

Returns the printed length

```
int pg_fieldptlen (int result_id, int row_number, string field_name)
```

Pg_fieldptlen() will return the actual printed length (number of characters) of a specific value in a PostgreSQL result. Row numbering starts at 0. This function will return -1 on an error.

pg_fieldsize (PHP 3, PHP 4)

Returns the internal storage size of the named field

```
int pg_fieldsize (int result_id, int field_number)
```

Pg_fieldsize() will return the internal storage size (in bytes) of the field number in the given PostgreSQL result. Field numbering starts at 0. A field size of -1 indicates a variable length field. This function will return false on error.

pg_fieldtype (PHP 3, PHP 4)

Returns the type name for the corresponding field number

```
string pg_fieldtype (int result_id, int field_number)
```

Pg_fieldtype() will return a string containing the type name of the given field in the given PostgreSQL result identifier. Field numbering starts at 0.

pg_freeresult (PHP 3, PHP 4)

Free result memory

```
int pg_freeresult (int result_id)
```

Pg_freeresult() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call **pg_freeresult()** with the result identifier as an argument and the associated result memory will be freed.

pg_getlastoid (PHP 3, PHP 4)

Returns the last object identifier

```
int pg_getlastoid (int result_id)
```

Pg_getlastoid() can be used to retrieve the `oid` assigned to an inserted tuple if the result identifier is used from the last command sent via **pg_exec()** and was an SQL INSERT. This function will return a positive integer if there was a valid `oid`. It will return -1 if an error occurred or the last command sent via **pg_exec()** was not an INSERT.

pg_host (PHP 3, PHP 4)

Returns the host name associated with the connection

```
string pg_host (int connection_id)
```

Pg_host() will return the host name of the given PostgreSQL connection identifier is connected to.

pg_loclose (PHP 3, PHP 4)

Close a large object

```
void pg_loclose (int fd)
```

Pg_loclose() closes an Inversion Large Object. `Fd` is a file descriptor for the large object from **pg_loopen()**.

pg_locreate (PHP 3, PHP 4)

Create a large object

```
int pg_locreate (int conn)
```

Pg_locreate() creates an Inversion Large Object and returns the *oid* of the large object. *conn* specifies a valid database connection. PostgreSQL access modes INV_READ, INV_WRITE, and INV_ARCHIVE are not supported, the object is created always with both read and write access. INV_ARCHIVE has been removed from PostgreSQL itself (version 6.3 and above).

pg_loexport (PHP 4 >= 4.0.1)

Export a large object to file

```
bool pg_loexport (int oid, int file [, int connection_id])
```

The *oid* argument specifies the object id of the large object to export and the *filename* argument specifies the pathname of the file. Returns FALSE if an error occurred, TRUE otherwise. Remember that handling large objects in PostgreSQL must happen inside a transaction.

pg_loimport (PHP 4 >= 4.0.1)

Import a large object from file

```
int pg_loimport (int file [, int connection_id])
```

The *filename* argument specifies the pathname of the file to be imported as a large object. Returns FALSE if an error occurred, object id of the just created large object otherwise. Remember that handling large objects in PostgreSQL must happen inside a transaction.

pg_loopen (PHP 3, PHP 4)

Open a large object

```
int pg_loopen (int conn, int objoid, string mode)
```

Pg_loopen() open an Inversion Large Object and returns file descriptor of the large object. The file descriptor encapsulates information about the connection. Do not close the connection before closing the large object file descriptor. *objoid* specifies a valid large object oid and *mode* can be either "r", "w", or "rw".

pg_loread (PHP 3, PHP 4)

Read a large object

```
string pg_loread (int fd, int len)
```

pg_loread() reads at most *len* bytes from a large object and returns it as a string. *fd* specifies a valid large object file descriptor and *len* specifies the maximum allowable size of the large object segment.

pg_loreadall (PHP 3, PHP 4)

Read a entire large object and send straight to browser

```
void pg_loreadall (int fd)
```

Pg_loreadall() reads a large object and passes it straight through to the browser after sending all pending headers. Mainly intended for sending binary data like images or sound.

pg_lounlink (PHP 3, PHP 4)

Delete a large object

```
void pg_lounlink (int conn, int lobjid)
```

Pg_lounlink() deletes a large object with the *lobjid* identifier for that large object.

pg_lowrite (PHP 3, PHP 4)

Write a large object

```
int pg_lowrite (int fd, string buf)
```

Pg_lowrite() writes at most to a large object from a variable *buf* and returns the number of bytes actually written, or false in the case of an error. *fd* is a file descriptor for the large object from **pg_loopen()**.

pg_numfields (PHP 3, PHP 4)

Returns the number of fields

```
int pg_numfields (int result_id)
```

Pg_numfields() will return the number of fields (columns) in a PostgreSQL result. The argument is a valid result identifier returned by **pg_exec()**. This function will return -1 on error.

pg_numrows (PHP 3, PHP 4)

Returns the number of rows

```
int pg_numrows (int result_id)
```

Pg_numrows() will return the number of rows in a PostgreSQL result. The argument is a valid result identifier returned by **pg_exec()**. This function will return -1 on error.

pg_options (PHP 3, PHP 4)

Get the options associated with the connection

```
string pg_options (int connection_id)
```

Pg_options() will return a string containing the options specified on the given PostgreSQL connection identifier.

pg_pconnect (PHP 3, PHP 4)

Open a persistant PostgreSQL connection

```
int pg_pconnect (string conn_string)
```

Returns a connection index on success, or false if the connection could not be made. Opens a connection to a PostgreSQL database. The arguments should be within a quoted string. The arguments available include *dbname*, *port*, *host*, *tty*, *options*, *user*, and *password*

This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple connections open at once.

The previous syntax of: `$conn = pg_pconnect ("host", "port", "options", "tty", "dbname")` has been deprecated.

See also **pg_connect()**.

pg_port (PHP 3, PHP 4)

Return the port number associated with the connection

```
int pg_port (int connection_id)
```

Pg_port() will return the port number that the given PostgreSQL connection identifier is connected to.

pg_put_line (PHP 4 >= 4.0.3)

Send a NULL-terminated string to PostgreSQL backend

```
bool pg_put_line ([resource connection_id, string data])
```

pg_put_line() sends a NULL-terminated string to the PostgreSQL backend server. This is useful for example for very high-speed inserting of data into a table, initiated by starting a PostgreSQL copy-operation. That final NULL-character is added automatically. Returns TRUE if successfull, FALSE otherwise.

Note: Note the application must explicitly send the two characters "\." on a final line to indicate to the backend that it has finished sending its data.

See also **pg_end_copy()**.

Example 1. High-speed insertion of data into a table

```
<?php
$conn = pg_pconnect ("dbname=foo");
pg_exec($conn, "create table bar (a int4, b char(16), d float8)");
pg_exec($conn, "copy bar from stdin");
pg_put_line($conn, "3\thello world\t4.5\n");
pg_put_line($conn, "4\tgoodbye world\t7.11\n");
pg_put_line($conn, "\\\.\n");
pg_end_copy($conn);
?>
```

pg_result (PHP 3, PHP 4)

Returns values from a result identifier

```
mixed pg_result (int result_id, int row_number, mixed fieldname)
```

Pg_result() will return values from a result identifier produced by **pg_Exec()**. The *row_number* and *fieldname* specify what cell in the table of results to return. Row numbering starts from 0. Instead of naming the field, you may use the field index as an unquoted number. Field indices start from 0.

PostgreSQL has many built in types and only the basic ones are directly supported here. All forms of integer, boolean and oid types are returned as integer values. All forms of float, and real types are returned as double values. All other types, including arrays are returned as strings formatted in the same default PostgreSQL manner that you would see in the **psql** program.

pg_set_client_encoding (PHP 4 >= 4.0.3)

Set the client encoding

```
int pg_set_client_encoding ([int connection, string encoding])
```

The function set the client encoding and return 0 if success or -1 if error.

encoding is the client encoding and can be either : SQL_ASCII, EUC_JP, EUC_CN, EUC_KR, EUC_TW, UNICODE, MULE_INTERNAL, LATINX (X=1...9), KOI8, WIN, ALT, SJIS, BIG5, WIN1250.

Note: This function requires PHP-4.0.2 or higher and PostgreSQL-7.0 or higher.

The function used to be called **pg_setclientencoding()**.

See also **pg_client_encoding()**.

pg_client_encoding (PHP 4 >= 4.0.3)

Get the client encoding

```
string pg_client_encoding ([int connection])
```

The function returns the client encoding as the string. The returned string should be either : SQL_ASCII, EUC_JP, EUC_CN, EUC_KR, EUC_TW, UNICODE, MULE_INTERNAL, LATINX (X=1...9), KOI8, WIN, ALT, SJIS, BIG5, WIN1250.

Note: This function requires PHP-4.0.2 or higher and PostgreSQL-7.0 or higher.

The function used to be called **pg_clientencoding()**.

See also **pg_set_client_encoding()**.

pg_trace (PHP 4 >= 4.0.1)

Enable tracing a PostgreSQL connection

```
bool pg_trace (string filename [, string mode [, int connection]])
```

Enables tracing of the PostgreSQL frontend/backend communication to a debugging file. To fully understand the results one needs to be familiar with the internals of PostgreSQL communication protocol. For those who are not, it can still be useful for tracing errors in queries sent to the server, you could do for example **grep '^To backend'** **trace.log** and see what query actually were sent to the PostgreSQL server.

Filename and *mode* are the same as in **fopen()** (*mode* defaults to 'w'), *connection* specifies the connection to trace and defaults to the last one opened.

Returns TRUE if *filename* could be opened for logging, FALSE otherwise.

See also **fopen()** and **pg_untrace()**.

pg_tty (PHP 3, PHP 4)

Return the tty name associated with the connection

```
string pg_tty (int connection_id)
```

Pg_tty() will return the tty name that server side debugging output is sent to on the given PostgreSQL connection identifier.

pg_untrace (PHP 4 >= 4.0.1)

Disable tracing of a PostgreSQL connection

```
bool pg_untrace ([int connection])
```

Stop tracing started by **pg_trace()**. *connection* specifies the connection that was traced and defaults to the last one opened.

Returns always TRUE.

See also **pg_trace()**.

LVI. Program Execution functions

escapeshellarg (PHP 4 >= 4.0.3)

escape a string to be used as a shell argument

```
string escapeshellarg (string arg)
```

EscapeShellArg() adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument. This function should be used to escape individual arguments to shell functions coming from user input. The shell functions include **exec()**, **system()** and the [backtick operator](#). A standard use would be:

```
system("ls ".EscapeShellArg($dir))
```

See also **exec()**, **popen()**, **system()**, and the [backtick operator](#).

escapeshellcmd (PHP 3, PHP 4)

escape shell metacharacters

```
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, or to the [backtick operator](#). A standard use would be:

```
$e = EscapeShellCmd($userinput);
system("echo $e"); // here we don't care if $e has spaces
$f = EscapeShellCmd($filename);
system("touch \"/tmp/$f\"; ls -l \"/tmp/$f\""); // and here we do, so we use quotes
```

See also **escapeshellarg()**, **exec()**, **popen()**, **system()**, and the [backtick operator](#).

exec (PHP 3, PHP 4)

Execute an external program

```
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.

Note also that if you start a program using this function and want to leave it running in the background, you have to make sure that the output of that program is redirected to a file or some other output stream or else PHP will hang until the execution of the program ends.

See also **system()**, **PassThru()**, **popen()**, **EscapeShellCmd()**, and the [backtick operator](#).

passthru (PHP 3, PHP 4)

Execute an external program and display raw output

```
void 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.

Note that if you start a program using this function and want to leave it running in the background, you have to make sure that the output of that program is redirected to a file or some other output stream or else PHP will hang until the execution of the program ends.

See also **exec()**, **system()**, **popen()**, **EscapeShellCmd()**, and the [backtick operator](#).

system (PHP 3, PHP 4)

Execute an external program and display output

```
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.

Note also that if you start a program using this function and want to leave it running in the background, you have to make sure that the output of that program is redirected to a file or some other output stream or else PHP will hang until the execution of the program ends.

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.

Returns the last line of the command output on success, and false on failure.

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 **exec()**, **PassThru()**, **popen()**, **EscapeShellCmd()**, and the [backtick operator](#).

LVII. Pspell Functions

These functions allow you to check the spelling of a word and offer suggestions.

You need the aspell and pspell libraries, available from <http://aspell.sourceforge.net/> and <http://pspell.sourceforge.net/> respectively, and add the `-with-pspell[=dir]` option when compiling php.

pspell_add_to_personal (PHP 4 >= 4.0.2)

Add the word to a personal wordlist.

```
int pspell_add_to_personal (int dictionary_link, string word)
```

Pspell_add_to_personal() adds a word to the personal wordlist. If you used **pspell_new_config()** with **pspell_config_personal()** to open the dictionary, you can save the wordlist later with **pspell_save_wordlist()**. Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

Example 1. Pspell_add_to_personal()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
$pspell_link = pspell_new_config ($pspell_config);

pspell_add_to_personal ($pspell_link, "Vlad");
pspell_save_wordlist ($pspell_link);
```

pspell_add_to_session (PHP 4 >= 4.0.2)

Add the word to the wordlist in the current session.

```
int pspell_add_to_session (int dictionary_link, string word)
```

Pspell_add_to_session() adds a word to the wordlist associated with the current session. It is very similar to **pspell_add_to_personal()**

pspell_check (PHP 4 >= 4.0.2)

Check a word

```
boolean pspell_check (int dictionary_link, string word)
```

Pspell_check() checks the spelling of a word and returns true if the spelling is correct, false if not.

Example 1. Pspell_check()

```
$pspell_link = pspell_new ("en");

if (pspell_check ($pspell_link, "testt")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

pspell_clear_session (PHP 4 >= 4.0.2)

Clear the current session.

```
int pspell_clear_session (int dictionary_link)
```

Pspell_clear_session() clears the current session. The current wordlist becomes blank, and, for example, if you try to save it with **pspell_save_wordlist()**, nothing happens.

Example 1. Pspell_add_to_personal()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
$pspell_link = pspell_new_config ($pspell_config);

pspell_add_to_personal ($pspell_link, "Vlad");
pspell_clear_session ($pspell_link);
pspell_save_wordlist ($pspell_link); // "Vlad" will not be saved
```

pspell_config_create (PHP 4 >= 4.0.2)

Create a config used to open a dictionary.

```
int pspell_config_create (string language [, string spelling [, string jargon [, string encoding]]])
```

Pspell_config_create() has a very similar syntax to **pspell_new()**. In fact, using **pspell_config_create()** immediately followed by **pspell_new_config()** will produce the exact same result. However, after creating a new config, you can also use **pspell_config_***() functions before calling **pspell_new_config()** to take advantage of some advanced functionality.

The language parameter is the language code which consists of the two letter ISO 639 language code and an optional two letter ISO 3166 country code after a dash or underscore.

The spelling parameter is the requested spelling for languages with more than one spelling such as English. Known values are 'american', 'british', and 'canadian'.

The jargon parameter contains extra information to distinguish two different words lists that have the same language and spelling parameters.

The encoding parameter is the encoding that words are expected to be in. Valid values are 'utf-8', 'iso8859-*', 'koi8-r', 'viscii', 'cp1252', 'machine unsigned 16', 'machine unsigned 32'. This parameter is largely untested, so be careful when using.

The mode parameter is the mode in which spellchecker will work. There are several modes available:

- PSPELL_FAST - Fast mode (least number of suggestions)
- PSPELL_NORMAL - Normal mode (more suggestions)
- PSPELL_BAD_SPELLERS - Slow mode (a lot of suggestions)

For more information and examples, check out inline manual pspell website:<http://pspell.sourceforge.net/>.

Example 1. Pspell_config_create()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
pspell_config_repl ($pspell_config, "/var/dictionaries/custom.repl");
$pspell_link = pspell_new_personal (pspell_config);
```

pspell_config_ignore (PHP 4 >= 4.0.2)

Ignore words less than N characters long.

```
int pspell_config_ignore (int dictionary_link, int n)
```

Pspell_config_ignore() should be used on a config before calling **pspell_new_config()**. This function allows short words to be skipped by the spellchecker. Words less then n characters will be skipped.

Example 1. Pspell_config_ignore()

```
$pspell_config = pspell_config_create ("en");
pspell_config_ignore($pspell_config, 5);
$pspell_link = pspell_new_config($pspell_config);
pspell_check($pspell_link, "abcd"); //will not result in an error
```

pspell_config_mode (PHP 4 >= 4.0.2)

Change the mode number of suggestions returned.

```
int pspell_config_mode (int dictionary_link, int mode)
```

Pspell_config_mode() should be used on a config before calling **pspell_new_config()**. This function determines how many suggestions will be returned by **pspell_suggest()**.

The mode parameter is the mode in which spellchecker will work. There are several modes available:

- PSPELL_FAST - Fast mode (least number of suggestions)
- PSPELL_NORMAL - Normal mode (more suggestions)
- PSPELL_BAD_SPELLERS - Slow mode (a lot of suggestions)

Example 1. Pspell_config_mode()

```
$pspell_config = pspell_config_create ("en");
pspell_config_mode($pspell_config, PSPELL_FAST);
$pspell_link = pspell_new_config($pspell_config);
pspell_check($pspell_link, "thecat");
```

pspell_config_personal (PHP 4 >= 4.0.2)

Set a file that contains personal wordlist.

```
int pspell_config_personal (int dictionary_link, string file)
```

Pspell_config_personal() should be used on a config before calling **pspell_new_config()**. The personal wordlist will be loaded and used in addition to the standard one after you call **pspell_new_config()**. If the file does not exist, it will be created. The file is also the file where **pspell_save_wordlist()** will save personal wordlist to. The file should be writable by whoever php runs as (e.g. nobody). Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

Example 1. Pspell_config_personal()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
$pspell_link = pspell_new_config ($pspell_config);
pspell_check ($pspell_link, "thecat");
```

pspell_config_repl (PHP 4 >= 4.0.2)

Set a file that contains replacement pairs.

```
int pspell_config_repl (int dictionary_link, string file)
```

Pspell_config_repl() should be used on a config before calling **pspell_new_config()**. The replacement pairs improve the quality of the spellchecker. When a word is misspelled, and a proper suggestion was not found in the list, **pspell_store_replacement()** can be used to store a replacement pair and then **pspell_save_wordlist()** to save the wordlist along with the replacement pairs. The file should be writable by whoever php runs as (e.g. nobody). Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

Example 1. Pspell_config_repl()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
pspell_config_repl ($pspell_config, "/var/dictionaries/custom.repl");
$pspell_link = pspell_new_config ($pspell_config);
pspell_check ($pspell_link, "thecat");
```

pspell_config_runtogther (PHP 4 >= 4.0.2)

Consider run-together words as valid compounds.

```
int pspell_config_runtogther (int dictionary_link, boolean flag)
```

Pspell_config_runtogether() should be used on a config before calling **pspell_new_config()**. This function determines whether run-together words will be treated as legal compounds. That is, "thecat" will be a legal compound, although there should be a space between the two words. Changing this setting only affects the results returned by **pspell_check()**; **pspell_suggest()** will still return suggestions.

Example 1. Pspell_config_runtogether()

```
$pspell_config = pspell_config_create ("en");
pspell_config_runtogether ($pspell_config, true);
$pspell_link = pspell_new_config ($pspell_config);
pspell_check ($pspell_link, "thecat");
```

pspell_config_save_repl (PHP 4 >= 4.0.2)

Determine whether to save a replacement pairs list along with the wordlist.

```
int pspell_config_save_repl (int dictionary_link, boolean flag)
```

Pspell_config_save_repl() should be used on a config before calling **pspell_new_config()**. It determines whether **pspell_save_wordlist()** will save the replacement pairs along with the wordlist. Usually there is no need to use this function because if **pspell_config_repl()** is used, the replacement pairs will be saved by **pspell_save_wordlist()** anyway, and if it is not, the replacement pairs will not be saved. Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

pspell_new (PHP 4 >= 4.0.2)

Load a new dictionary

```
int pspell_new (string language [, string spelling [, string jargon [, string encoding [, int mode]]]])
```

Pspell_new() opens up a new dictionary and returns the dictionary link identifier for use in other pspell functions.

The language parameter is the language code which consists of the two letter ISO 639 language code and an optional two letter ISO 3166 country code after a dash or underscore.

The spelling parameter is the requested spelling for languages with more than one spelling such as English. Known values are 'american', 'british', and 'canadian'.

The jargon parameter contains extra information to distinguish two different words lists that have the same language and spelling parameters.

The encoding parameter is the encoding that words are expected to be in. Valid values are 'utf-8', 'iso8859-*', 'koi8-r', 'viscii', 'cp1252', 'machine unsigned 16', 'machine unsigned 32'. This parameter is largely untested, so be careful when using.

The mode parameter is the mode in which spellchecker will work. There are several modes available:

- PSPELL_FAST - Fast mode (least number of suggestions)
- PSPELL_NORMAL - Normal mode (more suggestions)

- PSPELL_BAD_SPELLERS - Slow mode (a lot of suggestions)
- PSPELL_RUN_TOGETHER - Consider run-together words as legal compounds. That is, "thecat" will be a legal compound, although there should be a space between the two words. Changing this setting only affects the results returned by **pspell_check()**; **pspell_suggest()** will still return suggestions.

Mode is a bitmask constructed from different constants listed above. However, PSPELL_FAST, PSPELL_NORMAL and PSPELL_BAD_SPELLERS are mutually exclusive, so you should select only one of them.

For more information and examples, check out inline manual pspell website:<http://pspell.sourceforge.net/>.

Example 1. Pspell_new()

```
$pspell_link = pspell_new ("en", "", "", "",
                           (PSPELL_FAST|PSPELL_RUN_TOGETHER));
```

pspell_new_config (PHP 4 >= 4.0.2)

Load a new dictionary with settings based on a given config

```
int pspell_new_config (int config)
```

Pspell_new_config() opens up a new dictionary with settings specified in a config, created with **pspell_config_create()** and modified with **pspell_config_***() functions. This method provides you with the most flexibility and has all the functionality provided by **pspell_new()** and **pspell_new_personal()**.

The config parameter is the one returned by **pspell_config_create()** when the config was created.

Example 1. Pspell_new_config()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
pspell_config_repl ($pspell_config, "/var/dictionaries/custom.repl");
$pspell_link = pspell_new_personal (pspell_config);
```

pspell_new_personal (PHP 4 >= 4.0.2)

Load a new dictionary with personal wordlist

```
int pspell_new_personal (string personal, string language [, string spelling [, string
jargon [, string encoding [, int mode]]]])
```

Pspell_new_personal() opens up a new dictionary with a personal wordlist and returns the dictionary link identifier for use in other pspell functions. The wordlist can be modified and saved with **pspell_save_wordlist()**, if desired. However, the replacement pairs are not saved. In order to save replacement pairs, you should create a config using **pspell_config_create()**, set the personal wordlist file with **pspell_config_personal()**, set the file for replacement pairs with **pspell_config_repl()**, and open a new dictionary with **pspell_new_config()**.

The personal parameter specifies the file where words added to the personal list will be stored. It should be an absolute filename beginning with '/' because otherwise it will be relative to \$HOME, which is "/root" for most systems, and is probably not what you want.

The language parameter is the language code which consists of the two letter ISO 639 language code and an optional two letter ISO 3166 country code after a dash or underscore.

The spelling parameter is the requested spelling for languages with more than one spelling such as English. Known values are 'american', 'british', and 'canadian'.

The jargon parameter contains extra information to distinguish two different words lists that have the same language and spelling parameters.

The encoding parameter is the encoding that words are expected to be in. Valid values are 'utf-8', 'iso8859-*', 'koi8-r', 'viscii', 'cp1252', 'machine unsigned 16', 'machine unsigned 32'. This parameter is largely untested, so be careful when using.

The mode parameter is the mode in which spellchecker will work. There are several modes available:

- PSPELL_FAST - Fast mode (least number of suggestions)
- PSPELL_NORMAL - Normal mode (more suggestions)
- PSPELL_BAD_SPELLERS - Slow mode (a lot of suggestions)
- PSPELL_RUN_TOGETHER - Consider run-together words as legal compounds. That is, "thecat" will be a legal compound, although there should be a space between the two words. Changing this setting only affects the results returned by **pspell_check()**; **pspell_suggest()** will still return suggestions.

Mode is a bitmask constructed from different constants listed above. However, PSPELL_FAST, PSPELL_NORMAL and PSPELL_BAD_SPELLERS are mutually exclusive, so you should select only one of them.

For more information and examples, check out inline manual pspell website:<http://pspell.sourceforge.net/>.

Example 1. Pspell_new_personal()

```
$pspell_link = pspell_new_personal ( "/var/dictionaries/custom.pws",
"en", "", "", "", PSPELL_FAST|PSPELL_RUN_TOGETHER));
```

pspell_save_wordlist (PHP 4 >= 4.0.2)

Save the personal wordlist to a file.

```
int pspell_save_wordlist (int dictionary_link)
```

Pspell_save_wordlist() saves the personal wordlist from the current session. The dictionary has to be open with **pspell_new_personal()**, and the location of files to be saved specified with **pspell_config_personal()** and (optionally) **pspell_config_repl()**. Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

Example 1. Pspell_add_to_personal()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/tmp/dicts/newdict");
$pspell_link = pspell_new_config ($pspell_config);

pspell_add_to_personal ($pspell_link, "Vlad");
pspell_save_wordlist ($pspell_link);
```

pspell_store_replacement (PHP 4 >= 4.0.2)

Store a replacement pair for a word

```
int pspell_store_replacement (int dictionary_link, string misspelled, string correct)
```

Pspell_store_replacement() stores a replacement pair for a word, so that replacement can be returned by **pspell_suggest()** later. In order to be able to take advantage of this function, you have to use **pspell_new_personal()** to open the dictionary. In order to permanently save the replacement pair, you have to use **pspell_config_personal()** and **pspell_config_repl()** to set the path where to save your custom wordlists, and then use **pspell_save_wordlist()** for the changes to be written to disk. Please, note that this function will not work unless you have pspell .11.2 and aspell .32.5 or later.

Example 1. Pspell_store_replacement()

```
$pspell_config = pspell_config_create ("en");
pspell_config_personal ($pspell_config, "/var/dictionaries/custom.pws");
pspell_config_repl ($pspell_config, "/var/dictionaries/custom.repl");
$pspell_link = pspell_new_config ($pspell_config);

pspell_store_replacement ($pspell_link, $misspelled, $correct);
pspell_save_wordlist ($pspell_link);
```

pspell_suggest (PHP 4 >= 4.0.2)

Suggest spellings of a word

```
array pspell_suggest (int dictionary_link, string word)
```

Pspell_suggest() returns an array of possible spellings for the given word.

Example 1. Pspell_suggest()

```
$pspell_link = pspell_new ("en");

if (!pspell_check ($pspell_link, "testt")) {
    $suggestions = pspell_suggest ($pspell_link, "testt");

    for ($i=0; $i < count ($suggestions); $i++) {
        echo "Possible spelling: " . $suggestions[$i] . "<br>";
    }
}
```

LVIII. GNU Readline

The **readline()** functions implement an interface to the GNU Readline library. These are functions that provide editable command lines. An example being the way Bash allows you to use the arrow keys to insert characters or scroll through command history. Because of the interactive nature of this library, it will be of little use for writing Web applications, but may be useful when writing scripts meant to be run from a shell.

The home page of the GNU Readline project is <http://cnswww.cns.cwru.edu/~chet/readline/rlist.html>. It's maintained by Chet Ramey, who's also the author of Bash.

readline (PHP 4 >= 4.0b4)

Reads a line

```
string readline ([string prompt])
```

This function returns a single string from the user. You may specify a string with which to prompt the user. The line returned has the ending newline removed. You must add this line to the history yourself using **readline_add_history()**.

Example 1. Readline()

```
//get 3 commands from user
for ($i=0; $i < 3; $i++) {
    $line = readline ("Command: ");
    readline_add_history ($line);
}

//dump history
print_r (readline_list_history());

//dump variables
print_r (readline_info());
```

readline_add_history (PHP 4 >= 4.0b4)

Adds a line to the history

```
void readline_add_history (string line)
```

This function adds a line to the command line history.

readline_clear_history (PHP 4 >= 4.0b4)

Clears the history

```
boolean readline_clear_history (void )
```

This function clears the entire command line history.

readline_completion_function (PHP 4 >= 4.0b4)

Registers a completion function

```
boolean readline_completion_function (string line)
```

This function registers a completion function. You must supply the name of an existing function which accepts a partial command line and returns an array of possible matches. This is the same kind of functionality you'd get if you hit your tab key while using Bash.

readline_info (PHP 4 >= 4.0b4)

Gets/sets various internal readline variables

```
mixed readline_info ([string varname [, string newvalue]])
```

If called with no parameters, this function returns an array of values for all the setting readline uses. The elements will be indexed by the following values: done, end, erase_empty_line, library_version, line_buffer, mark, pending_input, point, prompt, readline_name, and terminal_name.

If called with one parameter, the value of that setting is returned. If called with two parameters, the setting will be changed to the given value.

readline_list_history (PHP 4 >= 4.0b4)

Lists the history

```
array readline_list_history (void )
```

This function returns an array of the entire command line history. The elements are indexed by integers starting at zero.

readline_read_history (PHP 4 >= 4.0b4)

Reads the history

```
boolean readline_read_history (string filename)
```

This function reads a command history from a file.

readline_write_history (PHP 4 >= 4.0b4)

Writes the history

```
boolean readline_write_history (string filename)
```

This function writes the command history to a file.

LIX. GNU Recode functions

This module contains an interface to the GNU Recode library, version 3.5. To be able to use the functions defined in this module you must compile your PHP interpreter using the `--with-recode` option. In order to do so, you must have GNU Recode 3.5 or higher installed on your system.

The GNU Recode library converts files between various coded character sets and surface encodings. When this cannot be achieved exactly, it may get rid of the offending characters or fall back on approximations. The library recognises or produces nearly 150 different character sets and is able to convert files between almost any pair. Most RFC 1345 character sets are supported.

recode_string (PHP 3>= 3.0.13, PHP 4 >= 4.0RC1)

Recode a string according to a recode request

```
string recode_string (string request, string string)
```

Recode the string *string* according to the recode request *request*. Returns the recoded string or FALSE, if unable to perform the recode request.

A simple recode request may be "lat1..iso646-de". See also the GNU Recode documentation of your installation for detailed instructions about recode requests.

Example 1. Basic recode_string() example:

```
print recode_string ("us..flat", "The following character has a diacritical mark: &aacute;");
```

recode (PHP 4 >= 4.0RC1)

Recode a string according to a recode request

```
string recode_string (string request, string string)
```

Note: This is an alias for **recode_string()**. It has been added in PHP 4.

recode_file (PHP 3>= 3.0.13, PHP 4 >= 4.0RC1)

Recode from file to file according to recode request

```
boolean recode_file (string request, resource input, resource output)
```

Recode the file referenced by file handle *input* into the file referenced by file handle *output* according to the recode *request*. Returns FALSE, if unable to comply, TRUE otherwise.

This function does not currently process filehandles referencing remote files (URLs). Both filehandles must refer to local files.

Example 1. Basic recode_file() example

```
$input = fopen ('input.txt', 'r');
$output = fopen ('output.txt', 'w');
recode_file ("us..flat", $input, $output);
```


LX. Regular Expression Functions (Perl-Compatible)

The syntax for patterns used in these functions closely resembles Perl. The expression should be enclosed in the delimiters, a forward slash (/), for example. Any character can be used for delimiter as long as it's not alphanumeric or backslash (\). If the delimiter character has to be used in the expression itself, it needs to be escaped by backslash. Since PHP 4.0.4, you can also use Perl-style (), {}, [], and <> matching delimiters.

The ending delimiter may be followed by various modifiers that affect the matching. See [Pattern Modifiers](#).

Example 1. Examples of valid patterns

- /<|\w+>/
- |(\d{3})-\d+|Sm
- /^(?i)php[34]/
- {^|s+(s+)?\$}

Example 2. Examples of invalid patterns

- /href='(.*)' - missing ending delimiter
- \w+\s*\w+/J - unknown modifier 'J'
- 1-\d3-\d3-\d4| - missing starting delimiter

Note: The Perl-compatible regular expression functions are available in PHP 4 and in PHP 3.0.9 and up.

preg_match (PHP 3>= 3.0.9, PHP 4)

Perform a regular expression match

```
int preg_match (string pattern, string subject [, array matches])
```

Searches *subject* for a match to the regular expression given in *pattern*.

If *matches* is provided, then it is filled with the results of search. \$matches[0] will contain the text that match the full pattern, \$matches[1] will have the text that matched the first captured parenthesized subpattern, and so on.

Returns true if a match for *pattern* was found in the subject string, or false if not match was found or an error occurred.

Example 1. Find the string of text "php"

```
// the "i" after the pattern delimiter indicates a case-insensitive search
if (preg_match ("/php/i", "PHP is the web scripting language of choice."))
{
    print "A match was found.";
} else {
    print "A match was not found.";
}
```

Example 2. find the word "web"

```
// the \b in the pattern indicates a word boundary, so only the distinct
// word "web" is matched, and not a word partial like "webbing" or "cobweb"
if (preg_match ("/\bweb\b/i", "PHP is the web scripting language of choice."))
{
    print "A match was found.";
} else {
    print "A match was not found.";
}
if (preg_match ("/\bweb\b/i", "PHP is the website scripting language of choice."))
{
    print "A match was found.";
} else {
    print "A match was not found.";
}
```

Example 3. Getting the domain name out of a URL

```
// get host name from URL
preg_match("/^(http://)?([^\/]*)/", "http://www.php.net/index.html", $matches);
$host = $matches[2];
// get last two segments of host name
preg_match("/[^\.\\/.]+\\.[^\\.\\/.]+$/i", $host, $matches);
echo "domain name is: ".$matches[0]."\n";
```

This example will produce:

```
domain name is: php.net
```

See also **preg_match_all()**, **preg_replace()**, and **preg_split()**.

preg_match_all (PHP 3>= 3.0.9, PHP 4)

Perform a global regular expression match

```
int preg_match_all (string pattern, string subject, array matches [, int order])
```

Searches *subject* for all matches to the regular expression given in *pattern* and puts them in *matches* in the order specified by *order*.

After the first match is found, the subsequent searches are continued on from end of the last match.

order can be one of two things:

PREG_PATTERN_ORDER

Orders results so that \$matches[0] is an array of full pattern matches, \$matches[1] is an array of strings matched by the first parenthesized subpattern, and so on.

```
preg_match_all ("|<[^>]+>(.*)</[^>]+>|U",
    "<b>example: </b><div align=left>this is a test</div>",
    $out, PREG_PATTERN_ORDER);
print $out[0][0].", ".$out[0][1]."\n";
print $out[1][0].", ".$out[1][1]."\n"
```

This example will produce:

```
<b>example: </b>, <div align=left>this is a test</div>
example: , this is a test
```

So, \$out[0] contains array of strings that matched full pattern, and \$out[1] contains array of strings enclosed by tags.

PREG_SET_ORDER

Orders results so that \$matches[0] is an array of first set of matches, \$matches[1] is an array of second set of matches, and so on.

```
preg_match_all ("|<[^>]+>(.*)</[^>]+>|U",
    "<b>example: </b><div align=left>this is a test</div>",
    $out, PREG_SET_ORDER);
print $out[0][0].", ".$out[0][1]."\n";
print $out[1][0].", ".$out[1][1]."\n"
```

This example will produce:

```
<b>example: </b>, example:
<div align=left>this is a test</div>, this is a test
```

In this case, \$matches[0] is the first set of matches, and \$matches[0][0] has text matched by full pattern, \$matches[0][1] has text matched by first subpattern and so on. Similarly, \$matches[1] is the second set of matches, etc.

If *order* is not specified, it is assumed to be PREG_PATTERN_ORDER.

Returns the number of full pattern matches, or false if no match is found or an error occurred.

Example 1. Getting all phone numbers out of some text.

```
preg_match_all ("/\((\d{3})\)\s+(\d{3})-(\d{4})/x",
    "Call 555-1212 or 1-800-555-1212", $phones);
```

Example 2. Find matching HTML tags (greedy)

```
// the \\2 is an example of backreferencing. This tells pcre that
// it must match the 2nd set of parenthesis in the regular expression
// itself, which would be the ([\w]+) in this case.
$html = "<b>bold text</b><a href=howdy.html>click me</a>

preg_match_all ("/(<([\w]+)[^>]*>)(.*)(<\/\\\2>)/", $html, $matches);

for ($i=0; $i < count($matches[0]); $i++) {
echo "matched: ".$matches[0][$i]."\n";
echo "part 1: ".$matches[1][$i]."\n";
echo "part 2: ".$matches[3][$i]."\n";
echo "part 3: ".$matches[4][$i]."\n\n";
```

This example will produce:

```
matched: <b>bold text</b>
part 1: <b>
part 2: bold text
part 3: </b>

matched: <a href=howdy.html>click me</a>
part 1: <a href=howdy.html>
part 2: click me
part 3: </a>
```

See also **preg_match()**, **preg_replace()**, and **preg_split()**.

preg_replace (PHP 3>= 3.0.9, PHP 4)

Perform a regular expression search and replace

```
mixed preg_replace (mixed pattern, mixed replacement, mixed subject [, int limit])
```

Searches *subject* for matches to *pattern* and replaces them with *replacement*. If *limit* is specified, then only *limit* matches will be replaced; if *limit* is omitted or is -1, then all matches are replaced.

Replacement may contain references of the form \\n or (since PHP 4.0.4) \$n, with the latter form being the preferred one. Every such reference will be replaced by the text captured by the n'th parenthesized pattern. n can be from 0 to 99, and \\0 or \$0 refers to the text matched by the whole pattern. Opening parentheses are counted from left to right (starting from 1) to obtain the number of the capturing subpattern.

If no matches are found in *subject*, then it will be returned unchanged.

Every parameter to **preg_replace()** can be an array.

If *subject* is an array, then the search and replace is performed on every entry of *subject*, and the return value is an array as well.

If *pattern* and *replacement* are arrays, then **preg_replace()** takes a value from each array and uses them to do search and replace on *subject*. If *replacement* has fewer values than *pattern*, then empty string is used for the

rest of replacement values. If *pattern* is an array and *replacement* is a string; then this replacement string is used for every value of *pattern*. The converse would not make sense, though.

/e modifier makes **preg_replace()** treat the *replacement* parameter as PHP code after the appropriate references substitution is done. Tip: make sure that *replacement* constitutes a valid PHP code string, otherwise PHP will complain about a parse error at the line containing **preg_replace()**.

/F modifier means that the *replacement* is taken to be a function name. This function will be called and passed an array of matched elements in the subject string. The function should return the replacement string. This modifier was added in PHP 4.0.4.

Example 1. Replacing several values

```
$patterns = array ("/(19|20)(\d{2})-(\d{1,2})-(\d{1,2})/",
    "/^\\s*{(\w+)}\\s*/=");
$replace = array ("\\3/\\4/\\1\\2", "{$\\1 =}");
print preg_replace ($patterns, $replace, "{startDate} = 1999-5-27");
```

This example will produce:

```
$startDate = 5/27/1999
```

Example 2. Using /e modifier

```
preg_replace ("/(</?)(\w+)([^>]*>)/e",
    "'\\1'.strtoupper('\\2').'\\3',
$html_body);
```

This would capitalize all HTML tags in the input text.

Example 3. Convert HTML to text

```
// $document should contain an HTML document.
// This will remove HTML tags, javascript sections
// and white space. It will also convert some
// common HTML entities to their text equivalent.

$search = array ('<script[^>]*?>.*?</script>'si', // Strip out javascript
    '<[/\!]*?[^<>]*?>'si', // Strip out html tags
    '([\r\n])[\s]+', // Strip out white space
    '&(quot|#34);'i', // Replace html entities
    '&(amp|#38);'i',
    '&(lt|#60);'i',
    '&(gt|#62);'i',
    '&(nbsp|#160);'i',
    '&(iexcl|#161);'i',
    '&(cent|#162);'i',
    '&(pound|#163);'i',
    '&(copy|#169);'i',
    '&#(\d+);'e"); // evaluate as php

$replace = array ("",
    "",
    "\\\1",
    "\\",
    "&",
    "<",
    ">",
```

```

" " ,
chr(161),
chr(162),
chr(163),
chr(169),
"chr(\\"1)" );

$text = preg_replace ($search, $replace, $document);

```

Note: Parameter *limit* was added after PHP 4.0.1pl2.

See also **preg_match()**, **preg_match_all()**, and **preg_split()**.

preg_split (PHP 3>= 3.0.9, PHP 4)

Split string by a regular expression

```
array preg_split (string pattern, string subject [, int limit [, int flags]])
```

Note: Parameter *flags* was added in PHP 4 Beta 3.

Returns an array containing substrings of *subject* split along boundaries matched by *pattern*.

If *limit* is specified, then only substrings up to *limit* are returned.

If *flags* is PREG_SPLIT_NO_EMPTY then only non-empty pieces will be returned by **preg_split()**.

Example 1. preg_split() example

Get the parts of a search string.

```
// split the phrase by any number of commas or space characters,
// which include " ", \r, \t, \n and \f
$keywords = preg_split ("/[\s,]+/", "hypertext language, programming");
```

Splitting a string into component characters.

```
$str = 'string';
$chars = preg_split ('///', $str, 0, PREG_SPLIT_NO_EMPTY);
print_r($chars);
```

See also **preg_match()**, **preg_match_all()**, and **preg_replace()**.

preg_quote (PHP 3>= 3.0.9, PHP 4)

Quote regular expression characters

```
string preg_quote (string str [, string delimiter])
```

preg_quote() takes *str* and puts a backslash in front of every character that is part of the regular expression syntax. This is useful if you have a run-time string that you need to match in some text and the string may contain special regex characters.

If the optional *delimiter* is specified, it will also be escaped. This is useful for escaping the delimiter that is required by the PCRE functions. The / is the most commonly used delimiter.

The special regular expression characters are:

```
. \ \ + * ? [ ^ ] $ ( ) { } = ! < > | :
```

Example 1.

```
$keywords = "$40 for a g3/400";
$keywords = preg_quote ($keywords, "/");
echo $keywords; // returns \$40 for a g3\400
```

Example 2. Italicizing a word within some text

```
// In this example, preg_quote($word) is used to keep the
// asterisks from having special meaning to the regular
// expression.

$textbody = "This book is *very* difficult to find.";
$word = "*very*";
$textbody = preg_replace ("/".preg_quote($word)."/",
                        "<i>".$word."</i>",
                        $textbody);
```

preg_grep (PHP 4)

Return array entries that match the pattern

```
array preg_grep (string pattern, array input)
```

preg_grep() returns the array consisting of the elements of the *input* array that match the given *pattern*.

Since PHP 4.0.4, the results returned by **preg_grep()** are indexed using the keys from the input array. If this behavior is undesirable, use **array_values()** on the array returned by **preg_grep()** to reindex the values.

Example 1. preg_grep() example

```
// return all array elements
// containing floating point numbers
$f1_array = preg_grep ('/^(\d+)?\.\d+$/ ', $array);
```

Pattern Modifiers

(unknown)

Describes possible modifiers in regex patterns

The current possible PCRE modifiers are listed below. The names in parentheses refer to internal PCRE names for these modifiers.

i (PCRE_CASELESS)

If this modifier is set, letters in the pattern match both upper and lower case letters.

m (PCRE_MULTILINE)

By default, PCRE treats the subject string as consisting of a single "line" of characters (even if it actually contains several newlines). The "start of line" metacharacter (^) matches only at the start of the string, while the "end of line" metacharacter (\$) matches only at the end of the string, or before a terminating newline (unless *E* modifier is set). This is the same as Perl.

When this modifier is set, the "start of line" and "end of line" constructs match immediately following or immediately before any newline in the subject string, respectively, as well as at the very start and end. This is equivalent to Perl's /m modifier. If there are no "\n" characters in a subject string, or no occurrences of ^ or \$ in a pattern, setting this modifier has no effect.

s (PCRE_DOTALL)

If this modifier is set, a dot metacharacter in the pattern matches all characters, including newlines. Without it, newlines are excluded. This modifier is equivalent to Perl's /s modifier. A negative class such as [^a] always matches a newline character, independent of the setting of this modifier.

x (PCRE_EXTENDED)

If this modifier is set, whitespace data characters in the pattern are totally ignored except when escaped or inside a character class, and characters between an unescaped # outside a character class and the next newline character, inclusive, are also ignored. This is equivalent to Perl's /x modifier, and makes it possible to include comments inside complicated patterns. Note, however, that this applies only to data characters. Whitespace characters may never appear within special character sequences in a pattern, for example within the sequence (? which introduces a conditional subpattern.

e

If this modifier is set, **preg_replace()** does normal substitution of backreferences in the replacement string, evaluates it as PHP code, and uses the result for replacing the search string. This modifier cannot be used along with /F modifier.

Only **preg_replace()** uses this modifier; it is ignored by other PCRE functions.

F

If this modifier is set, **preg_replace()** treats the replacement parameter as a function name that should be called to provide the replacement string. The function is passed an array of matched elements in the subject string. This modifier cannot be used along with /e modifier.

Only **preg_replace()** uses this modifier; it is ignored by other PCRE functions. This modifier was added in PHP 4.0.4.

A (PCRE_ANCHORED)

If this modifier is set, the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched (the "subject string"). This effect can also be achieved by appropriate constructs in the pattern itself, which is the only way to do it in Perl.

D (PCRE_DOLLAR_ENDONLY)

If this modifier is set, a dollar metacharacter in the pattern matches only at the end of the subject string. Without this modifier, a dollar also matches immediately before the final character if it is a newline (but not before any other newlines). This modifier is ignored if *m* modifier is set. There is no equivalent to this modifier in Perl.

S

When a pattern is going to be used several times, it is worth spending more time analyzing it in order to speed up the time taken for matching. If this modifier is set, then this extra analysis is performed. At present, studying a pattern is useful only for non-anchored patterns that do not have a single fixed starting character.

***U* (PCRE_UNGREEDY)**

This modifier inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?". It is not compatible with Perl. It can also be set by a (?U) modifier setting within the pattern.

***X* (PCRE_EXTRA)**

This modifier turns on additional functionality of PCRE that is incompatible with Perl. Any backslash in a pattern that is followed by a letter that has no special meaning causes an error, thus reserving these combinations for future expansion. By default, as in Perl, a backslash followed by a letter with no special meaning is treated as a literal. There are at present no other features controlled by this modifier.

Pattern Syntax (unknown)

Describes PCRE regex syntax

The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5, with just a few differences (see below). The current implementation corresponds to Perl 5.005.

The differences described here are with respect to Perl 5.005.

1. By default, a whitespace character is any character that the C library function isspace() recognizes, though it is possible to compile PCRE with alternative character type tables. Normally isspace() matches space, formfeed, newline, carriage return, horizontal tab, and vertical tab. Perl 5 no longer includes vertical tab in its set of whitespace characters. The \v escape that was in the Perl documentation for a long time was never in fact recognized. However, the character itself was treated as whitespace at least up to 5.002. In 5.004 and 5.005 it does not match \s.

2. PCRE does not allow repeat quantifiers on lookahead assertions. Perl permits them, but they do not mean what you might think. For example, (?!.a){3} does not assert that the next three characters are not "a". It just asserts that the next character is not "a" three times.

3. Capturing subpatterns that occur inside negative lookahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any such patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative lookahead assertion contains just one branch.

4. Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence "\0" can be used in the pattern to represent a binary zero.

5. The following Perl escape sequences are not supported: \l, \u, \L, \U, \E, \Q. In fact these are implemented by Perl's general string-handling and are not part of its pattern matching engine.

6. The Perl \G assertion is not supported as it is not relevant to single pattern matches.

7. Fairly obviously, PCRE does not support the (?{code}) construction.

8. There are at the time of writing some oddities in Perl 5.005_02 concerned with the settings of captured strings when part of a pattern is repeated. For example, matching "aba" against the pattern /^(a(b)?)+\$/ sets \$2 to the value "b", but matching "aabbaa" against /^(aa(bb)?)+\$/ leaves \$2 unset. However, if the pattern is changed to /^(aa(b(b))?)?+\$/ then \$2 (and \$3) get set.

In Perl 5.004 \$2 is set in both cases, and that is also true of PCRE. If in the future Perl changes to a consistent state that is different, PCRE may change to follow.

9. Another as yet unresolved discrepancy is that in Perl 5.005_02 the pattern /^(a)?(?:1)a|b)+\$/ matches the string "a", whereas in PCRE it does not. However, in both Perl and PCRE /^(a)?a/ matched against "a" leaves \$1 unset.

10. PCRE provides some extensions to the Perl regular expression facilities:

(a) Although lookbehind assertions must match fixed length strings, each alternative branch of a lookbehind assertion can match a different length of string. Perl 5.005 requires them all to have the same length.

(b) If PCRE_DOLLAR_ENDONLY is set and PCRE_MULTILINE is not set, the \$ meta-character matches only at the very end of the string.

(c) If PCRE_EXTRA is set, a backslash followed by a letter with no special meaning is faulted.

(d) If PCRE_UNGREEDY is set, the greediness of the repetition quantifiers is inverted, that is, by default they are not greedy, but if followed by a question mark they are.

The syntax and semantics of the regular expressions sup-

ported by PCRE are described below. Regular expressions are also described in the Perl documentation and in a number of other books, some of which have copious examples. Jeffrey Friedl's "Mastering Regular Expressions", published by O'Reilly (ISBN 1-56592-257-3), covers them in great detail. The description here is intended as reference documentation.

A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject. As a trivial example, the pattern

The quick brown fox

matches a portion of a subject string that is identical to itself. The power of regular expressions comes from the ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of *meta-characters*, which do not stand for themselves but instead are interpreted in some special way.

There are two different sets of meta-characters: those that are recognized anywhere in the pattern except within square brackets, and those that are recognized in square brackets. Outside square brackets, the meta-characters are as follows:

- \ general escape character with several uses
- ^ assert start of subject (or line, in multiline mode)
- \$ assert end of subject (or line, in multiline mode)
- . match any character except newline (by default)
- [start character class definition
- | start of alternative branch
- (start subpattern
-) end subpattern
- ? extends the meaning of (
- also 0 or 1 quantifier
- also quantifier minimizer
- * 0 or more quantifier
- + 1 or more quantifier
- { start min/max quantifier

Part of a pattern that is in square brackets is called a "character class". In a character class the only meta-characters are:

- \ general escape character
- ^ negate the class, but only if the first character
- indicates character range
-] terminates the character class

The following sections describe the use of each of the meta-characters.

BACKSLASH

The backslash character has several uses. Firstly, if it is followed by a non-alphabetic character, it takes away any special meaning that character may have. This use of backslash as an escape character applies both inside and outside character classes.

For example, if you want to match a "*" character, you write "*" in the pattern. This applies whether or not the following character would otherwise be interpreted as a meta-character, so it is always safe to precede a non-alphabetic with "\" to specify that it stands for itself. In particular, if you want to match a backslash, you write "\\".

If a pattern is compiled with the PCRE_EXTENDED option, whitespace in the pattern (other than in a character class) and characters between a "#" outside a character class and the next newline character are ignored. An escaping backslash can be used to include a whitespace or "#" character as part of the pattern.

A second use of backslash provides a way of encoding non-printing characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern, but when a pattern is being prepared by text editing, it is usually easier to use one of the following escape sequences than the binary character it represents:

- \a alarm, that is, the BEL character (hex 07)
- \cx "control-x", where x is any character
- \e escape (hex 1B)
- \f formfeed (hex 0C)
- \n newline (hex 0A)
- \r carriage return (hex 0D)
- \t tab (hex 09)
- \xhh character with hex code hh
- \ddd character with octal code ddd, or backreference

The precise effect of "\cx" is as follows: if "x" is a lower case letter, it is converted to upper case. Then bit 6 of the character (hex 40) is inverted. Thus "\cz" becomes hex 1A, but "\c{" becomes hex 3B, while "\c;" becomes hex 7B.

After "\x", up to two hexadecimal digits are read (letters can be in upper or lower case).

After "\0" up to two further octal digits are read. In both cases, if there are fewer than two digits, just those that are present are used. Thus the sequence "\0\x\07" specifies two binary zeros followed by a BEL character. Make sure you supply two digits after the initial zero if the character that follows is itself an octal digit.

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, PCRE reads it

and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing left parentheses in the expression, the entire sequence is taken as a *back reference*. A description of how this works is given later, following the discussion of parenthesized subpatterns.

Inside a character class, or if the decimal number is greater than 9 and there have not been that many capturing subpatterns, PCRE re-reads up to three octal digits following the backslash, and generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves. For example:

- \040 is another way of writing a space
- \40 is the same, provided there are fewer than 40 previous capturing subpatterns
- \7 is always a back reference
- \11 might be a back reference, or another way of writing a tab
- \011 is always a tab
- \0113 is a tab followed by the character "3"
- \113 is the character with octal code 113 (since there can be no more than 99 back references)
- \377 is a byte consisting entirely of 1 bits
- \81 is either a back reference, or a binary zero followed by the two characters "8" and "1"

Note that octal values of 100 or greater must not be introduced by a leading zero, because no more than three octal digits are ever read.

All the sequences that define a single byte value can be used both inside and outside character classes. In addition, inside a character class, the sequence "\b" is interpreted as the backspace character (hex 08). Outside a character class it has a different meaning (see below).

The third use of backslash is for specifying generic character types:

- \d any decimal digit
- \D any character that is not a decimal digit
- \s any whitespace character
- \S any character that is not a whitespace character
- \w any "word" character
- \W any "non-word" character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

A "word" character is any letter or digit or the underscore character, that is, any character which can be part of a Perl "word". The definition of letters and digits is controlled by PCRE's character tables, and may vary if locale-

specific matching is taking place (see "Locale support" above). For example, in the "fr" (French) locale, some character codes greater than 128 are used for accented letters, and these are matched by \w.

These character type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

The fourth use of backslash is for certain simple assertions. An assertion specifies a condition that has to be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The backslashed assertions are

- \b word boundary
- \B not a word boundary
- \A start of subject (independent of multiline mode)
- \Z end of subject or newline at end (independent of multiline mode)
- \z end of subject (independent of multiline mode)

These assertions may not appear in character classes (but note that "\b" has a different meaning, namely the backspace character, inside a character class).

A word boundary is a position in the subject string where the current character and the previous character do not both match \w or \W (i.e. one matches \w and the other matches \W), or the start or end of the string if the first or last character matches \w, respectively.

The \A, \Z, and \z assertions differ from the traditional circumflex and dollar (described below) in that they only ever match at the very start and end of the subject string, whatever options are set. They are not affected by the PCRE_NOTBOL or PCRE_NOTEOL options. The difference between \Z and \z is that \Z matches before a newline that is the last character of the string as well as at the end of the string, whereas \z matches only at the end.

CIRCUMFLEX AND DOLLAR

Outside a character class, in the default matching mode, the circumflex character is an assertion which is true only if the current matching point is at the start of the subject string. Inside a character class, circumflex has an entirely different meaning (see below).

Circumflex need not be the first character of the pattern if

a number of alternatives are involved, but it should be the first thing in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex, that is, if the pattern is constrained to match only at the start of the subject, it is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character is an assertion which is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar need not be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

The meaning of dollar can be changed so that it matches only at the very end of the string, by setting the PCRE_DOLLAR_ENDONLY option at compile or matching time. This does not affect the \Z assertion.

The meanings of the circumflex and dollar characters are changed if the PCRE_MULTILINE option is set. When this is the case, they match immediately after and immediately before an internal "\n" character, respectively, in addition to matching at the start and end of the subject string. For example, the pattern `/^abc$/` matches the subject string "def\nabc" in multiline mode, but not otherwise. Consequently, patterns that are anchored in single line mode because all branches start with "^" are not anchored in multiline mode. The PCRE_DOLLAR_ENDONLY option is ignored if PCRE_MULTILINE is set.

Note that the sequences \A, \Z, and \z can be used to match the start and end of the subject in both modes, and if all branches of a pattern start with \A is it always anchored, whether PCRE_MULTILINE is set or not.

FULL STOP (PERIOD, DOT)

Outside a character class, a dot in the pattern matches any one character in the subject, including a non-printing character, but not (by default) newline. If the PCRE_DOTALL option is set, then dots match newlines as well. The handling of dot is entirely independent of the handling of circumflex and dollar, the only relationship being that they both involve newline characters. Dot has no special meaning in a character class.

SQUARE BRACKETS

An opening square bracket introduces a character class, terminated by a closing square bracket. A closing square

bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject; the character must be in the set of characters defined by the class, unless the first character in the class is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class [aeiou] matches any lower case vowel, while [^aeiou] matches any character that is not a lower case vowel. Note that a circumflex is just a convenient notation for specifying the characters which are in the class by enumerating those that are not. It is not an assertion: it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

When caseless matching is set, any letters in a class represent both their upper case and lower case versions, so for example, a caseless [aeiou] matches "A" as well as "a", and a caseless [^aeiou] does not match "A", whereas a caseful version would.

The newline character is never treated in any special way in character classes, whatever the setting of the PCRE_DOTALL or PCRE_MULTILINE options is. A class such as [^a] will always match a newline.

The minus (hyphen) character can be used to specify a range of characters in a character class. For example, [d-m] matches any letter between d and m, inclusive. If a minus character is required in a class, it must be escaped with a backslash or appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class.

It is not possible to have the literal character "]" as the end character of a range. A pattern such as [W-]46] is interpreted as a class of two characters ("W" and "-") followed by a literal string "46]", so it would match "W46]" or "-46]". However, if the "]" is escaped with a backslash it is interpreted as the end of range, so [W-\]46] is interpreted as a single class containing a range followed by two separate characters. The octal or hexadecimal representation of "]" can also be used to end a range.

Ranges operate in ASCII collating sequence. They can also be used for characters specified numerically, for example [\000-\037]. If a range that includes letters is used when caseless matching is set, it matches the letters in either case. For example, [W-c] is equivalent to [][\^_`wxyzabc],

matched caselessly, and if character tables for the "fr" locale are in use, [\xc8-\xcb] matches accented E characters in both cases.

The character types \d, \D, \s, \S, \w, and \W may also appear in a character class, and add the characters that they match to the class. For example, [\dABCDEF] matches any hexadecimal digit. A circumflex can conveniently be used with the upper case character types to specify a more restricted set of characters than the matching lower case type. For example, the class [^\W_] matches any letter or digit, but not underscore.

All non-alphabetic characters other than \, -, ^ (at the start) and the terminating] are non-special in character classes, but it does no harm if they are escaped.

VERTICAL BAR

Vertical bar characters are used to separate alternative patterns. For example, the pattern

```
gilbert|sullivan
```

matches either "gilbert" or "sullivan". Any number of alternatives may appear, and an empty alternative is permitted (matching the empty string). The matching process tries each alternative in turn, from left to right, and the first one that succeeds is used. If the alternatives are within a subpattern (defined below), "succeeds" means matching the rest of the main pattern as well as the alternative in the subpattern.

INTERNAL OPTION SETTING

The settings of PCRE_CASELESS, PCRE_MULTILINE, PCRE_DOTALL, and PCRE_EXTENDED can be changed from within the pattern by a sequence of Perl option letters enclosed between "(" and ")". The option letters are

- i for PCRE_CASELESS
- m for PCRE_MULTILINE
- s for PCRE_DOTALL
- x for PCRE_EXTENDED

For example, (?im) sets caseless, multiline matching. It is also possible to unset these options by preceding the letter with a hyphen, and a combined setting and unsetting such as (?im-sx), which sets PCRE_CASELESS and PCRE_MULTILINE while unsetting PCRE_DOTALL and PCRE_EXTENDED, is also permitted. If a letter appears both before and after the hyphen, the option is unset.

The scope of these option changes depends on where in the

pattern the setting occurs. For settings that are outside any subpattern (defined below), the effect is the same as if the options were set or unset at the start of matching. The following patterns all behave in exactly the same way:

```
(?i)abc
a(?i)bc
ab(?i)c
abc(?i)
```

which in turn is the same as compiling the pattern abc with PCRE_CASELESS set. In other words, such "top level" settings apply to the whole pattern (unless there are other changes inside subpatterns). If there is more than one setting of the same option at top level, the rightmost setting is used.

If an option change occurs inside a subpattern, the effect is different. This is a change of behaviour in Perl 5.005. An option change inside a subpattern affects only that part of the subpattern that follows it, so

```
(a(?i)b)c
```

matches abc and aBc and no other strings (assuming PCRE_CASELESS is not used). By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example,

```
(a(?i)b|c)
```

matches "ab", "aB", "c", and "C", even though when matching "C" the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time. There would be some very weird behaviour otherwise.

The PCRE-specific options PCRE_UNGREEDY and PCRE_EXTRA can be changed in the same way as the Perl-compatible options by using the characters U and X respectively. The (?X) flag setting is special in that it must always occur earlier in the pattern than any of the additional features it turns on, even when it is at top level. It is best put at the start.

SUBPATTERNS

Subpatterns are delimited by parentheses (round brackets), which can be nested. Marking part of a pattern as a subpattern does two things:

1. It localizes a set of alternatives. For example, the pattern

```
cat(aract|erpillar|)
```

matches one of the words "cat", "cataract", or "caterpillar". Without the parentheses, it would match "cataract", "erpillar" or the empty string.

2. It sets up the subpattern as a capturing subpattern (as defined above). When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller via the *ovector* argument of **pcre_exec()**. Opening parentheses are counted from left to right (starting from 1) to obtain the numbers of the capturing subpatterns.

For example, if the string "the red king" is matched against the pattern

```
the ((red|white) (king|queen))
```

the captured substrings are "red king", "red", and "king", and are numbered 1, 2, and 3.

The fact that plain parentheses fulfil two functions is not always helpful. There are often times when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by "?:", the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string "the white queen" is matched against the pattern

```
the ((?:red|white) (king|queen))
```

the captured substrings are "white queen" and "queen", and are numbered 1 and 2. The maximum number of captured substrings is 99, and the maximum number of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters may appear between the "?" and the ":". Thus the two patterns

```
(?:saturday|sunday)
(?::(?i)saturday|sunday)
```

match exactly the same set of strings. Because alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached, an option setting in one branch does affect subsequent branches, so the above patterns match "SUNDAY" as well as "Saturday".

REPETITION

Repetition is specified by quantifiers, which can follow any

of the following items:

- a single character, possibly escaped
- the . metacharacter
- a character class
- a back reference (see next section)
- a parenthesized subpattern (unless it is an assertion - see below)

The general repetition quantifier specifies a minimum and maximum number of permitted matches, by giving the two numbers in curly brackets (braces), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example:

`z{2,4}`

matches "zz", "zzz", or "zzzz". A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit; if the second number and the comma are both omitted, the quantifier specifies an exact number of required matches. Thus

`[aeiou]{3,}`

matches at least 3 successive vowels, but may match many more, while

`\d{8}`

matches exactly 8 digits. An opening curly bracket that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, `{,6}` is not a quantifier, but a literal string of four characters.

The quantifier `{0}` is permitted, causing the expression to behave as if the previous item and the quantifier were not present.

For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

- * is equivalent to `{0,}`
- + is equivalent to `{1,}`
- ? is equivalent to `{0,1}`

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit, for example:

`(a?)*`

Earlier versions of Perl and PCRE used to give an error at compile time for such patterns. However, because there are cases where this can be useful, such patterns are now

accepted, but if any repetition of the subpattern does in fact match no characters, the loop is forcibly broken.

By default, the quantifiers are "greedy", that is, they match as much as possible (up to the maximum number of permitted times), without causing the rest of the pattern to fail. The classic example of where this gives problems is in trying to match comments in C programs. These appear between the sequences /* and */ and within the sequence, individual * and / characters may appear. An attempt to match C comments by applying the pattern

```
\/*.*\*/
```

to the string

```
/* first command */ not comment /* second comment */
```

fails, because it matches the entire string due to the greediness of the .* item.

However, if a quantifier is followed by a question mark, then it ceases to be greedy, and instead matches the minimum number of times possible, so the pattern

```
\/*.*?/*
```

does the right thing with the C comments. The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled, as in

```
\d??\d
```

which matches one digit by preference, but can match two if that is the only way the rest of the pattern matches.

If the PCRE_UNGREEDY option is set (an option which is not available in Perl) then the quantifiers are not greedy by default, but individual ones can be made greedy by following them with a question mark. In other words, it inverts the default behaviour.

When a parenthesized subpattern is quantified with a minimum repeat count that is greater than 1 or with a limited maximum, more store is required for the compiled pattern, in proportion to the size of the minimum or maximum.

If a pattern starts with .* or .{0,} and the PCRE_DOTALL option (equivalent to Perl's /s) is set, thus allowing the . to match newlines, then the pattern is implicitly anchored, because whatever follows will be tried against every character position in the subject string, so there is no point in retrying the overall match at any position after the first.

PCRE treats such a pattern as though it were preceded by \A. In cases where it is known that the subject string contains no newlines, it is worth setting PCRE_DOTALL when the pattern begins with .* in order to obtain this optimization, or alternatively using ^ to indicate anchoring explicitly.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after

```
(tweedle[dume]{3}\s*)+
```

has matched "tweedledum tweedledee" the value of the captured substring is "tweedledee". However, if there are nested capturing subpatterns, the corresponding captured values may have been set in previous iterations. For example, after

```
/(a|(b))+/
```

matches "aba" the value of the second captured substring is "b".

BACK REFERENCES

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (i.e. to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. See the section entitled "Backslash" above for further details of the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. So the pattern

```
(sens|respons)e and \1ibility
```

matches "sense and sensibility" and "response and responsibility", but not "sense and responsibility". If caseful matching is in force at the time of the back reference, then the case of letters is relevant. For example,

```
((?i)rah)\s+\1
```

matches "rah rah" and "RAH RAH", but not "RAH rah", even though the original capturing subpattern is matched case-

lessly.

There may be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, then any back references to it always fail. For example, the pattern

`(a|(bc))\2`

always fails if it starts to match "a" rather than "bc". Because there may be up to 99 back references, all digits following the backslash are taken as part of a potential back reference number. If the pattern continues with a digit character, then some delimiter must be used to terminate the back reference. If the PCRE_EXTENDED option is set, this can be whitespace. Otherwise an empty comment can be used.

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used, so, for example, `(a\1)` never matches. However, such references can be useful inside repeated subpatterns. For example, the pattern

`(a|b\1)+`

matches any number of "a"s and also "aba", "ababaa" etc. At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. In order for this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of zero.

ASSERTIONS

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as `\b`, `\B`, `\A`, `\Z`, `\z`, `^` and `$` are described above. More complicated assertions are coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed. Lookahead assertions start with `(?=` for positive assertions and `(?!?` for negative assertions. For example,

`\w+(?=:;)`

matches a word followed by a semicolon, but does not include the semicolon in the match, and

`foo(?!bar)`

matches any occurrence of "foo" that is not followed by "bar". Note that the apparently similar pattern

```
(?!foo)bar
```

does not find an occurrence of "bar" that is preceded by something other than "foo"; it finds any occurrence of "bar" whatsoever, because the assertion `(?!foo)` is always true when the next three characters are "bar". A lookbehind assertion is needed to achieve this effect.

Lookbehind assertions start with `(?<=` for positive assertions and `(?<!` for negative assertions. For example,

```
(?<!foo)bar
```

does find an occurrence of "bar" that is not preceded by "foo". The contents of a lookbehind assertion are restricted such that all the strings it matches must have a fixed length. However, if there are several alternatives, they do not all have to have the same fixed length. Thus

```
(?<=bullock|donkey)
```

is permitted, but

```
(?<!dogs?|cats?)
```

causes an error at compile time. Branches that match different length strings are permitted only at the top level of a lookbehind assertion. This is an extension compared with Perl 5.005, which requires all branches to match the same length of string. An assertion such as

```
(?<=ab(c|de))
```

is not permitted, because its single top-level branch can match two different lengths, but it is acceptable if rewritten to use two top-level branches:

```
(?<=abc|abde)
```

The implementation of lookbehind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail. Lookbehinds in conjunction with once-only subpatterns can be particularly useful for matching at the ends of strings; an example is given at the end of the section on once-only subpatterns.

Several assertions (of any sort) may occur in succession. For example,

```
(?<=\d{3})(?<!999)foo
```

matches "foo" preceded by three digits that are not "999". Notice that each of the assertions is applied independently at the same point in the subject string. First there is a check that the previous three characters are all digits, then there is a check that the same three characters are not "999". This pattern does not match "foo" preceded by six characters, the first of which are digits and the last three of which are not "999". For example, it doesn't match "123abcfoo". A pattern to do that is

```
(?=<\d{3}...)(?<!999)foo
```

This time the first assertion looks at the preceding six characters, checking that the first three are digits, and then the second assertion checks that the preceding three characters are not "999".

Assertions can be nested in any combination. For example,

```
(?=<(?<!foo)bar)baz
```

matches an occurrence of "baz" that is preceded by "bar" which in turn is not preceded by "foo", while

```
(?=<\d{3}(?!999)...)foo
```

is another pattern which matches "foo" preceded by three digits and any three characters that are not "999".

Assertion subpatterns are not capturing subpatterns, and may not be repeated, because it makes no sense to assert the same thing several times. If any kind of assertion contains capturing subpatterns within it, these are counted for the purposes of numbering the capturing subpatterns in the whole pattern. However, substring capturing is carried out only for positive assertions, because it does not make sense for negative assertions.

Assertions count towards the maximum of 200 parenthesized subpatterns.

ONCE-ONLY SUBPATTERNS

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to change the nature of the match, or to cause it fail earlier than it otherwise might, when the author of the pattern knows there is no point in carrying on.

Consider, for example, the pattern `\d+foo` when applied to the subject line

123456bar

After matching all 6 digits and then failing to match "foo", the normal action of the matcher is to try again with only 5 digits matching the \d+ item, and then with 4, and so on, before ultimately failing. Once-only subpatterns provide the means for specifying that once a portion of the pattern has matched, it is not to be re-evaluated in this way, so the matcher would give up immediately on failing to match "foo" the first time. The notation is another kind of special parenthesis, starting with (?> as in this example:

(?>\d+)bar

This kind of parenthesis "locks up" the part of the pattern it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern would match, if anchored at the current point in the subject string.

Once-only subpatterns are not capturing subpatterns. Simple cases such as the above example can be thought of as a maximizing repeat that must swallow everything it can. So, while both \d+ and \d+? are prepared to adjust the number of digits they match in order to make the rest of the pattern match, (?>\d+) can only match an entire sequence of digits.

This construction can of course contain arbitrarily complicated subpatterns, and it can be nested.

Once-only subpatterns can be used in conjunction with look-behind assertions to specify efficient matching at the end of the subject string. Consider a simple pattern such as

abcd\$

when applied to a long string which does not match. Because matching proceeds from left to right, PCRE will look for each "a" in the subject and then see if what follows matches the rest of the pattern. If the pattern is specified as

^.*abcd\$

then the initial .* matches the entire string at first, but when this fails (because there is no following "a"), it backtracks to match all but the last character, then all but the last two characters, and so on. Once again the search for "a" covers the entire string, from right to left, so we are no better off. However, if the pattern is written as

^(?>.*)(?=<=abcd)

then there can be no backtracking for the `.*` item; it can match only the entire string. The subsequent lookbehind assertion does a single test on the last four characters. If it fails, the match fails immediately. For long strings, this approach makes a significant difference to the processing time.

When a pattern contains an unlimited repeat inside a subpattern that can itself be repeated an unlimited number of times, the use of a once-only subpattern is the only way to avoid some failing matches taking a very long time indeed. The pattern

```
(\D+|<\d+>)*[!?]
```

matches an unlimited number of substrings that either consist of non-digits, or digits enclosed in `<>`, followed by either `!` or `?`. When it matches, it runs quickly. However, if it is applied to

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

it takes a long time before reporting failure. This is because the string can be divided between the two repeats in a large number of ways, and all have to be tried. (The example used `[!?]` rather than a single character at the end, because both PCRE and Perl have an optimization that allows for fast failure when a single character is used. They remember the last single character that is required for a match, and fail early if it is not present in the string.)

If the pattern is changed to

```
((?>|\D+)|<\d+>)*[!?]
```

sequences of non-digits cannot be broken, and failure happens quickly.

CONDITIONAL SUBPATTERNS

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern are

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)
```

If the condition is satisfied, the yes-pattern is used; otherwise the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are two kinds of condition. If the text between the parentheses consists of a sequence of digits, then the condition is satisfied if the capturing subpattern of that

number has previously matched. Consider the following pattern, which contains non-significant white space to make it more readable (assume the PCRE_EXTENDED option) and to divide it into three parts for ease of discussion:

```
(\()? [^()]+ (?(1)\))
```

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern that tests whether the first set of parentheses matched or not. If they did, that is, if subject started with an opening parenthesis, the condition is true, and so the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses.

If the condition is not a sequence of digits, it must be an assertion. This may be a positive or negative lookahead or lookbehind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line:

```
(?(?=^[a-z]*[a-z])  
 \d{2}-[a-z]{3}-\d{2} | \d{2}-\d{2}-\d{2})
```

The condition is a positive lookahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms dd-aaa-dd or dd-dd-dd, where aaa are letters and dd are digits.

COMMENTS

The sequence (?# marks the start of a comment which continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching at all.

If the PCRE_EXTENDED option is set, an unescaped # character outside a character class introduces a comment that continues up to the next newline character in the pattern.

RECURSIVE PATTERNS

Consider the problem of matching a string in parentheses, allowing for unlimited nested parentheses. Without the use of recursion, the best that can be done is to use a pattern

that matches up to some fixed depth of nesting. It is not possible to handle an arbitrary nesting depth. Perl 5.6 has provided an experimental facility that allows regular expressions to recurse (amongst other things). The special item (?R) is provided for the specific case of recursion. This PCRE pattern solves the parentheses problem (assume the PCRE_EXTENDED option is set so that white space is ignored):

```
\(( ((?>[^()]+) | (?R))* )\)
```

First it matches an opening parenthesis. Then it matches any number of substrings which can either be a sequence of non-parentheses, or a recursive match of the pattern itself (i.e. a correctly parenthesized substring). Finally there is a closing parenthesis.

This particular example pattern contains nested unlimited repeats, and so the use of a once-only subpattern for matching strings of non-parentheses is important when applying the pattern to strings that do not match. For example, when it is applied to

```
(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa()
```

it yields "no match" quickly. However, if a once-only subpattern is not used, the match runs for a very long time indeed because there are so many different ways the + and * repeats can carve up the subject, and all have to be tested before failure can be reported.

The values set for any capturing subpatterns are those from the outermost level of the recursion at which the subpattern value is set. If the pattern above is matched against

```
(ab(cd)ef)
```

the value for the capturing parentheses is "ef", which is the last value taken on at the top level. If additional parentheses are added, giving

```
\(( (( ((?>[^()]+) | (?R))* ) )\)
^           ^
          ^ then the string they capture
```

is "ab(cd)ef", the contents of the top level parentheses. If there are more than 15 capturing parentheses in a pattern, PCRE has to obtain extra memory to store data during a recursion, which it does by using `pcre_malloc`, freeing it via `pcre_free` afterwards. If no memory can be obtained, it saves data for the first 15 capturing parentheses only, as there is no way to give an out-of-memory error from within a recursion.

Certain items that may appear in patterns are more efficient than others. It is more efficient to use a character class like [aeiou] than a set of alternatives such as (a|e|i|o|u).

In general, the simplest construction that provides the required behaviour is usually the most efficient. Jeffrey Friedl's book contains a lot of discussion about optimizing regular expressions for efficient performance.

When a pattern begins with `.*` and the PCRE_DOTALL option is set, the pattern is implicitly anchored by PCRE, since it can match only at the start of a subject string. However, if PCRE_DOTALL is not set, PCRE cannot make this optimization, because the `.` metacharacter does not then match a newline, and if the subject string contains newlines, the pattern may match from the character immediately following one of them instead of from the very start. For example, the pattern

`(.*) second`

matches the subject "first\nand second" (where `\n` stands for a newline character) with the first captured substring being "and". In order to do this, PCRE has to retry the match starting after every newline in the subject.

If you are using such a pattern with subject strings that do not contain newlines, the best performance is obtained by setting PCRE_DOTALL, or starting the pattern with `^.*` to indicate explicit anchoring. That saves PCRE from having to scan along the subject looking for a newline to restart at.

Beware of patterns that contain nested indefinite repeats. These can take a long time to run when applied to a string that does not match. Consider the pattern fragment

`(a+)*`

This can match "aaaa" in 33 different ways, and this number increases very rapidly as the string gets longer. (The `*` repeat can match 0, 1, 2, 3, or 4 times, and for each of those cases other than 0, the `+` repeats can match different numbers of times.) When the remainder of the pattern is such that the entire match is going to fail, PCRE has in principle to try every possible variation, and this can take an extremely long time.

An optimization catches some of the more simple cases such as

`(a+)*b`

where a literal character follows. Before embarking on the standard matching procedure, PCRE checks that there is a "b" later in the subject string, and if there is not, it fails the match immediately. However, when there is no following literal this optimization cannot be used. You can see the difference by comparing the behaviour of

(a+)*\d

with the pattern above. The former gives a failure almost instantly when applied to a whole line of "a" characters, whereas the latter takes an appreciable time with strings longer than about 20 characters.

LXI. Regular Expression Functions (POSIX Extended)

Regular expressions are used for complex string manipulation in PHP. The functions that support regular expressions are:

- **ereg()**
- **ereg_replace()**
- **eregi()**
- **eregi_replace()**
- **split()**
- **spliti()**

These functions all take a regular expression string as their first argument. PHP uses the POSIX extended regular expressions as defined by POSIX 1003.2. For a full description of POSIX regular expressions see the regex man pages included in the regex directory in the PHP distribution. It's in manpage format, so you'll want to do something along the lines of **man /usr/local/src/regex/regex.7** in order to read it.

Example 1. Regular Expression Examples

```
ereg ("abc", $string);
/* Returns true if "abc"
   is found anywhere in $string. */

ereg ("^abc", $string);
/* Returns true if "abc"
   is found at the beginning of $string. */

ereg ("abc$", $string);
/* Returns true if "abc"
   is found at the end of $string. */

eregi ("(ozilla.[23]|MSIE.3)", $HTTP_USER_AGENT);
/* Returns true if client browser
   is Netscape 2, 3 or MSIE 3. */

ereg ("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)", $string,$regs);
/* Places three space separated words
   into $regs[1], $regs[2] and $regs[3]. */

$string = ereg_replace ("^", "<BR>", $string);
/* Put a <BR> tag at the beginning of $string. */

$string = ereg_replace ("\$", "<BR>", $string);
/* Put a <BR> tag at the end of $string. */

$string = ereg_replace ("\n", "", $string);
/* Get rid of any newline
   characters in $string. */
```


ereg (PHP 3, PHP 4)

Regular expression match

```
int ereg (string pattern, string string [, array regs])
```

Searches a *string* for matches to the regular expression given in *pattern*.

If matches are found for parenthesized substrings of *pattern* and the function is called with the third argument *regs*, the matches will be stored in the elements of the array *regs*. \$regs[1] will contain the substring which starts at the first left parenthesis; \$regs[2] will contain the substring starting at the second, and so on. \$regs[0] will contain a copy of *string*.

If **ereg()** finds any matches at all, \$regs will be filled with exactly ten elements, even though more or fewer than ten parenthesized substrings may actually have matched. This has no effect on **ereg()**'s ability to match more substrings. If no matches are found, \$regs will not be altered by **ereg()**.

Searching is case sensitive.

Returns true if a match for *pattern* was found in *string*, or false if no matches were found or an error occurred.

The following code snippet takes a date in ISO format (YYYY-MM-DD) and prints it in DD.MM.YYYY format:

Example 1. Ereg() Example

```
if (ereg ("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
    echo "$regs[3].$regs[2].$regs[1]";
} else {
    echo "Invalid date format: $date";
}
```

See also **eregi()**, **ereg_replace()**, and **eregi_replace()**.

ereg_replace (PHP 3, PHP 4)

Replace regular expression

```
string ereg_replace (string pattern, string replacement, string string)
```

This function scans *string* for matches to *pattern*, then replaces the matched text with *replacement*.

The modified string is returned. (Which may mean that the original string is returned if there are no matches to be replaced.)

If *pattern* contains parenthesized substrings, *replacement* may contain substrings of the form `\d`, which will be replaced by the text matching the digit'th parenthesized substring; `\0` will produce the entire contents of *string*. Up to nine substrings may be used. Parentheses may be nested, in which case they are counted by the opening parenthesis.

If no matches are found in *string*, then *string* will be returned unchanged.

For example, the following code snippet prints "This was a test" three times:

Example 1. Ereg_replace() Example

```
$string = "This is a test";
echo ereg_replace (" is", " was", $string);
echo ereg_replace ("( )is", "\1was", $string);
echo ereg_replace ("(( )is)", "\2was", $string);
```

One thing to take note of is that if you use an integer value as the *replacement* parameter, you may not get the results you expect. This is because **ereg_replace()** will interpret the number as the ordinal value of a character, and apply that. For instance:

Example 2. `ereg_replace()` Example

```
<?php
/* This will not work as expected. */
$num = 4;
$string = "This string has four words.";
$string = ereg_replace('four', $num, $string);
echo $string; /* Output: 'This string has    words.' */

/* This will work. */
$num = '4';
$string = "This string has four words.";
$string = ereg_replace('four', $num, $string);
echo $string; /* Output: 'This string has 4 words.' */
?>
```

See also **ereg()**, **ereg()**, and **ereg_replace()**.

eregi (PHP 3, PHP 4)

case insensitive regular expression match

```
int eregi (string pattern, string string [, array regs])
```

This function is identical to **ereg()** except that this ignores case distinction when matching alphabetic characters.

See also **ereg()**, **ereg_replace()**, and **ereg_replace()**.

ereg_replace (PHP 3, PHP 4)

replace regular expression case insensitive

```
string ereg_replace (string pattern, string replacement, string string)
```

This function is identical to **ereg_replace()** except that this ignores case distinction when matching alphabetic characters.

See also **ereg()**, **ereg()**, and **ereg_replace()**.

split (PHP 3, PHP 4)

split string into array by regular expression

```
array split (string pattern, string string [, int limit])
```

Returns an array of strings, each of which is a substring of *string* formed by splitting it on boundaries formed by the regular expression *pattern*. If *limit* is set, the returned array will contain a maximum of *limit* elements with the last element containing the whole rest of *string*. If an error occurs, **split()** returns false.

To split off the first four fields from a line from /etc/passwd:

Example 1. Split() Example

```
$passwd_list = split (":", $passwd_line, 5);
```

To parse a date which may be delimited with slashes, dots, or hyphens:

Example 2. Split() Example

```
$date = "04/30/1973"; // Delimiters may be slash, dot, or hyphen
list ($month, $day, $year) = split ('[/.-]', $date);
echo "Month: $month; Day: $day; Year: $year<br>\n";
```

Note that *pattern* is case-sensitive.

Note that if you don't require the power of regular expressions, it is faster to use **explode()**, which doesn't incur the overhead of the regular expression engine.

For users looking for a way to emulate perl's **\$chars = split(' ', \$str)** behaviour, please see the examples for **preg_split()**.

Please note that *pattern* is a regular expression. If you want to split on any of the characters which are considered special by regular expressions, you'll need to escape them first. If you think **split()** (or any other regex function, for that matter) is doing something weird, please read the file *regex.7*, included in the *regex/* subdirectory of the PHP distribution. It's in manpage format, so you'll want to do something along the lines of **man /usr/local/src/regex/regex.7** in order to read it.

See also: **spliti()**, **explode()**, and **implode()**.

spliti (PHP 4 >= 4.0.1)

Split string into array by regular expression case insensitive

```
array split (string pattern, string string [, int limit])
```

This function is identical to **split()** except that this ignores case distinction when matching alphabetic characters.

See also: **split()**, **explode()**, and **implode()**.

sql_regcase (PHP 3, PHP 4)

Make regular expression for case insensitive match

```
string sql_regcase (string string)
```

Returns a valid regular expression which will match *string*, ignoring case. This expression is *string* with each character converted to a bracket expression; this bracket expression contains that character's uppercase and lowercase form if applicable, otherwise it contains the original character twice.

Example 1. Sql_regcase() Example

```
echo sql_regcase ("Foo bar");
```

prints

```
[Ff][Oo][Oo] [Bb][Aa][Rr]
```

This can be used to achieve case insensitive pattern matching in products which support only case sensitive regular expressions.

LXII. Satellite CORBA client extension

The Satellite extension is used for accessing CORBA objects. You will need to set the `idl_directory=` entry in `php.ini` to a path where you store all IDL files you use.

OrbitObject (unknown)

Access CORBA objects

```
new OrbitObject (string ior)
```

This class provides access to a CORBA object. The *ior* parameter should be a string containing the IOR (Interoperable Object Reference) that identifies the remote object.

Example 1. Sample IDL file

```
interface MyInterface {
    void SetInfo (string info);
    string GetInfo();

    attribute int value;
}
```

Example 2. PHP code for accessing MyInterface

```
<?php
$obj = new OrbitObject ($ior);

$obj->SetInfo ("A 2GooD object");

echo $obj->GetInfo();

$obj->value = 42;

echo $obj->value;
?>
```

OrbitEnum (unknown)

Use CORBA enums

```
new OrbitEnum (string id)
```

This class represents the enumeration identified with the *id* parameter. The *id* can be either the name of the enumeration (e.g "MyEnum"), or the full repository id (e.g. "IDL:MyEnum:1.0").

Example 1. Sample IDL file

```
enum MyEnum {
    a,b,c,d,e
};
```

Example 2. PHP code for accessing MyEnum

```
<?php
$enum = new OrbitEnum ("MyEnum");

echo $enum->a; /* write 0 */
echo $enum->c; /* write 2 */
echo $enum->e; /* write 4 */
?>
```

OrbitStruct (unknown)

Use CORBA structs

```
new OrbitStruct (string id)
```

This class represents the structure identified with the *id* parameter. The *id* can be either the name of the struct (e.g "MyStruct"), or the full repository id (e.g. "IDL:MyStruct:1.0").

Example 1. Sample IDL file

```
struct MyStruct {
    short shortvalue;
    string stringvalue;
};

interface SomeInterface {
    void SetValues (MyStruct values);
    MyStruct GetValues();
}
```

Example 2. PHP code for accessing MyStruct

```
<?php
$obj = new OrbitObject ($ior);

$initial_values = new OrbitStruct ("IDL:MyStruct:1.0");
$initial_values->shortvalue = 42;
$initial_values->stringvalue = "HGTIG";

$obj->SetValues ($initial_values);

$values = $obj->GetValues();

echo $values->shortvalue;
echo $values->stringvalue;
?>
```

satellite_caught_exception (PHP 4 >= 4.0.3)

See if an exception was caught from the previous function

```
bool satellite_caught_exception ()
```

This function returns true if an exception has been caught.

Example 1. Sample IDL file

```
/* +++++++ Out of Cheese Error. Redo From Start. */
exception OutOfCheeseError {
    int parameter;
}

interface AnotherInterface {
    void AskWhy() raises (OutOfCheeseError);
}
```

Example 2. PHP code for handling CORBA exceptions

```
<?php
$obj = new OrbitObject ($ior);

$obj->AskWhy();

if (satellite_caught_exception()) {
    if ("IDL:OutOfCheeseError:1.0" == satellite_exception_id()) {
        $exception = satellite_exception_value();
        echo $exception->parameter;
    }
}
?>
```

satellite_exception_id (PHP 4 >= 4.0.3)

Get the repository id for the latest exception.

```
string satellite_exception_id ()
```

Return a repository id string. (E.g. "IDL:MyException:1.0".) For example usage see **satellite_caught_exception()**.

satellite_exception_value (PHP 4 >= 4.0.3)

Get the exception struct for the latest exception

```
OrbitStruct satellite_exception_value ()
```

Return an exception struct. For example usage see **satellite_caught_exception()**.

LXIII. Semaphore and Shared Memory Functions

This module provides semaphore functions using System V semaphores. Semaphores may be used to provide exclusive access to resources on the current machine, or to limit the number of processes that may simultaneously use a resource.

This module provides also shared memory functions using System V shared memory. Shared memory may be used to provide access to global variables. Different httpd-daemons and even other programs (such as Perl, C, ...) are able to access this data to provide a global data-exchange. Remember, that shared memory is NOT safe against simultaneous access. Use semaphores for synchronization.

Table 1. Limits of Shared Memory by the Unix OS

SHMMAX	max size of shared memory, normally 131072 bytes
SHMMIN	minimum size of shared memory, normally 1 byte
SHMMNI	max amount of shared memory segments on a system, normally 100
SHMSEG	max amount of shared memory segments per process, normally 6

Note: These functions do not work on Windows systems.

sem_get (PHP 3>= 3.0.6, PHP 4)

Get a semaphore id

```
int sem_get (int key [, int max_acquire [, int perm]])
```

Returns: A positive semaphore identifier on success, or false on error.

Sem_get() returns an id that can be used to access the System V semaphore with the given key. The semaphore is created if necessary using the permission bits specified in perm (defaults to 0666). The number of processes that can acquire the semaphore simultaneously is set to max_acquire (defaults to 1). Actually this value is set only if the process finds it is the only process currently attached to the semaphore.

A second call to **sem_get()** for the same key will return a different semaphore identifier, but both identifiers access the same underlying semaphore.

See also: **sem_acquire()** and **sem_release()**.

Note: This function does not work on Windows systems

sem_acquire (PHP 3>= 3.0.6, PHP 4)

Acquire a semaphore

```
int sem_acquire (int sem_identifier)
```

Returns: true on success, false on error.

Sem_acquire() blocks (if necessary) until the semaphore can be acquired. A process attempting to acquire a semaphore which it has already acquired will block forever if acquiring the semaphore would cause its max_acquire value to be exceeded.

After processing a request, any semaphores acquired by the process but not explicitly released will be released automatically and a warning will be generated.

See also: **sem_get()** and **sem_release()**.

sem_release (PHP 3>= 3.0.6, PHP 4)

Release a semaphore

```
int sem_release (int sem_identifier)
```

Returns: true on success, false on error.

Sem_release() releases the semaphore if it is currently acquired by the calling process, otherwise a warning is generated.

After releasing the semaphore, **sem_acquire()** may be called to re-acquire it.

See also: **sem_get()** and **sem_acquire()**.

Note: This function does not work on Windows systems

shm_attach (PHP 3>= 3.0.6, PHP 4)

Creates or open a shared memory segment

```
int shm_attach (int key [, int memsize [, int perm]])
```

Shm_attach() returns an id that can be used to access the System V shared memory with the given key, the first call creates the shared memory segment with mem_size (default: sysvshm.init_mem in the [configuration file](#), otherwise 10000 bytes) and the optional perm-bits (default: 0666).

A second call to **shm_attach()** for the same *key* will return a different shared memory identifier, but both identifiers access the same underlying shared memory. *Memsize* and *perm* will be ignored.

Note: This function does not work on Windows systems

shm_detach (PHP 3>= 3.0.6, PHP 4)

Disconnects from shared memory segment

```
int shm_detach (int shm_identifier)
```

Shm_detach() disconnects from the shared memory given by the *shm_identifier* created by **shm_attach()**. Remember, that shared memory still exist in the Unix system and the data is still present.

shm_remove (PHP 3>= 3.0.6, PHP 4)

Removes shared memory from Unix systems

```
int shm_remove (int shm_identifier)
```

Removes shared memory from Unix systems. All data will be destroyed.

Note: This function does not work on Windows systems

shm_put_var (PHP 3>= 3.0.6, PHP 4)

Inserts or updates a variable in shared memory

```
int shm_put_var (int shm_identifier, int variable_key, mixed variable)
```

Inserts or updates a *variable* with a given *variable_key*. All variable-types (double, int, string, array) are supported.

Note: This function does not work on Windows systems

shm_get_var (PHP 3>= 3.0.6, PHP 4)

Returns a variable from shared memory

```
mixed shm_get_var (int id, int variable_key)
```

Shm_get_var() returns the variable with a given *variable_key*. The variable is still present in the shared memory.

Note: This function does not work on Windows systems

shm_remove_var (PHP 3>= 3.0.6, PHP 4)

Removes a variable from shared memory

```
int shm_remove_var (int id, int variable_key)
```

Removes a variable with a given *variable_key* and frees the occupied memory.

Note: This function does not work on Windows systems

LXIV. SESAM database functions

SESAM/SQL-Server is a mainframe database system, developed by Fujitsu Siemens Computers, Germany. It runs on high-end mainframe servers using the operating system BS2000/OSD.

In numerous productive BS2000 installations, SESAM/SQL-Server has proven ...

- the ease of use of Java-, Web- and client/server connectivity,
- the capability to work with an availability of more than 99.99%,
- the ability to manage tens and even hundreds of thousands of users.

Now there is a PHP3 SESAM interface available which allows database operations via PHP-scripts.

Configuration notes: There is no standalone support for the PHP SESAM interface, it works only as an integrated Apache module. In the Apache PHP module, this [SESAM interface is configured](#) using Apache directives.

Table 1. SESAM Configuration directives

Directive	Meaning
php3_sesam_oml	Name of BS2000 PLAM library containing the loadable SESAM driver modules. Required for using SESAM functions. Example: php3_sesam_oml \$.SYSLNK.SESAM-SQL.030
php3_sesam_configfile	Name of SESAM application configuration file. Required for using SESAM functions. Example: php3_sesam_configfile \$SESAM.SESAM.CONF.AW It will usually contain a configuration like (see SESAM reference manual): CNF=B NAM=K NOTYPE
php3_sesam_messagecatalog	Name of SESAM message catalog file. In most cases, this directive is not necessary. Only if the SESAM message file is not installed in the system's BS2000 message file table, it can be set with this directive. Example: php3_sesam_messagecatalog \$.SYSMES.SESAM-SQL.030

In addition to the configuration of the PHP/SESAM interface, you have to configure the SESAM-Database server itself on your mainframe as usual. That means:

- starting the SESAM database handler (DBH), and
- connecting the databases with the SESAM database handler

To get a connection between a PHP script and the database handler, the `CNF` and `NAM` parameters of the selected SESAM configuration file must match the id of the started database handler.

In case of distributed databases you have to start a SESAM/SQL-DCN agent with the distribution table including the host and database names.

The communication between PHP (running in the POSIX subsystem) and the database handler (running outside the POSIX subsystem) is realized by a special driver module called SQLSCI and SESAM connection modules using

common memory. Because of the common memory access, and because PHP is a static part of the web server, database accesses are very fast, as they do not require remote accesses via ODBC, JDBC or UTM.

Only a small stub loader (SESMOD) is linked with PHP, and the SESAM connection modules are pulled in from SESAM's OML PLAM library. In the [configuration](#), you must tell PHP the name of this PLAM library, and the file link to use for the SESAM configuration file (As of SESAM V3.0, SQLSCI is available in the SESAM Tool Library, which is part of the standard distribution).

Because the SQL command quoting for single quotes uses duplicated single quotes (as opposed to a single quote preceded by a backslash, used in some other databases), it is advisable to set the PHP configuration directives `php3_magic_quotes_gpc` and `php3_magic_quotes_sybase` to `on` for all PHP scripts using the SESAM interface.

Runtime considerations: Because of limitations of the BS2000 process model, the driver can be loaded only after the Apache server has forked off its server child processes. This will slightly slow down the initial SESAM request of each child, but subsequent accesses will respond at full speed.

When explicitly defining a Message Catalog for SESAM, that catalog will be loaded each time the driver is loaded (i.e., at the initial SESAM request). The BS2000 operating system prints a message after successful load of the message catalog, which will be sent to Apache's `error_log` file. BS2000 currently does not allow suppression of this message, it will slowly fill up the log.

Make sure that the SESAM OML PLAM library and SESAM configuration file are readable by the user id running the web server. Otherwise, the server will be unable to load the driver, and will not allow to call any SESAM functions. Also, access to the database must be granted to the user id under which the Apache server is running. Otherwise, connections to the SESAM database handler will fail.

Cursor Types: The result cursors which are allocated for SQL "select type" queries can be either "sequential" or "scrollable". Because of the larger memory overhead needed by "scrollable" cursors, the default is "sequential".

When using "scrollable" cursors, the cursor can be freely positioned on the result set. For each "scrollable" query, there are global default values for the scrolling type (initialized to: `SESAM_SEEK_NEXT`) and the scrolling offset which can either be set once by `sesam_seek_row()` or each time when fetching a row using `sesam_fetch_row()`. When fetching a row using a "scrollable" cursor, the following post-processing is done for the global default values for the scrolling type and scrolling offset:

Table 2. Scrolled Cursor Post-Processing

Scroll Type	Action
<code>SESAM_SEEK_NEXT</code>	none
<code>SESAM_SEEK_PRIOR</code>	none
<code>SESAM_SEEK_FIRST</code>	set scroll type to <code>SESAM_SEEK_NEXT</code>
<code>SESAM_SEEK_LAST</code>	set scroll type to <code>SESAM_SEEK_PRIOR</code>
<code>SESAM_SEEK_ABSOLUTE</code>	Auto-Increment internal offset value
<code>SESAM_SEEK_RELATIVE</code>	none. (maintain global default <code>offset value</code> , which allows for, e.g., fetching each 10th row backwards)

Porting note: Because in the PHP world it is natural to start indexes at zero (rather than 1), some adaptions have been made to the SESAM interface: whenever an indexed array is starting with index 1 in the native SESAM interface, the PHP interface uses index 0 as a starting point. E.g., when retrieving columns with `sesam_fetch_row()`, the first column has the index 0, and the subsequent columns have indexes up to (but not including) the column count (`$array["count"]`). When porting SESAM applications from other high level languages to PHP, be aware of this changed interface. Where appropriate, the description of the respective php sesam functions include a note that the index is zero based.

Security concerns: When allowing access to the SESAM databases, the web server user should only have as little

privileges as possible. For most databases, only read access privilege should be granted. Depending on your usage scenario, add more access rights as you see fit. Never allow full control to any database for any user from the 'net! Restrict access to php scripts which must administer the database by using password control and/or SSL security.

Migration from other SQL databases: No two SQL dialects are ever 100% compatible. When porting SQL applications from other database interfaces to SESAM, some adaption may be required. The following typical differences should be noted:

- Vendor specific data types

Some vendor specific data types may have to be replaced by standard SQL data types (e.g., TEXT could be replaced by VARCHAR(max. size)).

- Keywords as SQL identifiers

In SESAM (as in standard SQL), such identifiers must be enclosed in double quotes (or renamed).

- Display length in data types

SESAM data types have a precision, not a display length. Instead of `int(4)` (intended use: integers up to '9999'), SESAM requires simply `int` for an implied size of 31 bits. Also, the only datetime data types available in SESAM are: DATE, TIME(3) and TIMESTAMP(3).

- SQL types with vendor-specific `unsigned`, `zerofill`, or `auto_increment` attributes

`Unsigned` and `zerofill` are not supported. `Auto_increment` is automatic (use `"INSERT ... VALUES(*, ...)"` instead of `"... VALUES(0, ...)"` to take advantage of SESAM-implied auto-increment).

- **int ... DEFAULT '0000'**

Numeric variables must not be initialized with string constants. Use **DEFAULT 0** instead. To initialize variables of the datetime SQL data types, the initialization string must be prefixed with the respective type keyword, as in: `CREATE TABLE exmpl (xtime timestamp(3) DEFAULT TIMESTAMP '1970-01-01 00:00:00.000' NOT NULL);`

- **\$count = xxxx_num_rows();**

Some databases promise to guess/estimate the number of the rows in a query result, even though the returned value is grossly incorrect. SESAM does not know the number of rows in a query result before actually fetching them. If you REALLY need the count, try **SELECT COUNT(...)** WHERE ..., it will tell you the number of hits. A second query will (hopefully) return the results.

- **DROP TABLE thename;**

In SESAM, in the **DROP TABLE** command, the table name must be either followed by the keyword **RESTRICT** or **CASCADE**. When specifying **RESTRICT**, an error is returned if there are dependent objects (e.g., VIEWS), while with **CASCADE**, dependent objects will be deleted along with the specified table.

Notes on the use of various SQL types: SESAM does not currently support the BLOB type. A future version of SESAM will have support for BLOB.

At the PHP interface, the following type conversions are automatically applied when retrieving SQL fields:

Table 3. SQL to PHP Type Conversions

SQL Type	PHP Type
SMALLINT, INTEGER	"integer"
NUMERIC, DECIMAL, FLOAT, REAL, DOUBLE	"double"
DATE, TIME, TIMESTAMP	"string"
VARCHAR, CHARACTER	"string"

When retrieving a complete row, the result is returned as an array. Empty fields are not filled in, so you will have to check for the existence of the individual fields yourself (use `isset()` or `empty()` to test for empty fields). That allows more user control over the appearance of empty fields (than in the case of an empty string as the representation of an empty field).

Support of SESAM's "multiple fields" feature: The special "multiple fields" feature of SESAM allows a column to consist of an array of fields. Such a "multiple field" column can be created like this:

Example 1. Creating a "multiple field" column

```
CREATE TABLE multi_field_test (
    pkey CHAR(20) PRIMARY KEY,
    multi(3) CHAR(12)
)
```

and can be filled in using:

Example 2. Filling a "multiple field" column

```
INSERT INTO multi_field_test (pkey, multi(2..3) )
    VALUES ('Second', '<first_val>', 'second_val'>)
```

Note that (like in this case) leading empty sub-fields are ignored, and the filled-in values are collapsed, so that in the above example the result will appear as `multi(1..2)` instead of `multi(2..3)`.

When retrieving a result row, "multiple columns" are accessed like "inlined" additional columns. In the example above, "pkey" will have the index 0, and the three "multi(1..3)" columns will be accessible as indices 1 through 3.

For specific SESAM details, please refer to the SESAM/SQL-Server documentation (english) (http://its.siemens.de/lobs/its/techinf/oltp/sesam/manuals/index_en.htm) or the SESAM/SQL-Server documentation (german) (http://its.siemens.de/lobs/its/techinf/oltp/sesam/manuals/index_gr.htm), both available online, or use the respective manuals.

sesam_connect (unknown)

Open SESAM database connection

```
boolean sesam_connect (string catalog, string schema, string user)
```

Returns TRUE if a connection to the SESAM database was made, or FALSE on error.

sesam_connect() establishes a connection to an SESAM database handler task. The connection is always "persistant" in the sense that only the very first invocation will actually load the driver from the configured SESAM OML PLAM library. Subsequent calls will reuse the driver and will immediately use the given catalog, schema, and user.

When creating a database, the "*catalog*" name is specified in the SESAM configuration directive
//ADD-SQL-DATABASE-CATALOG-LIST ENTRY-1 = *CATALOG(CATALOG-NAME = catalogname,...)

The "*schema*" references the desired database schema (see SESAM handbook).

The "*user*" argument references one of the users which are allowed to access this "*catalog*" / "*schema*" combination. Note that "*user*" is completely independent from both the system's user id's and from HTTP user/password protection. It appears in the SESAM configuration only.

See also **sesam_disconnect()**.

Example 1. Connect to a SESAM database

```
<?php
if (!sesam_connect ("mycatalog", "myschema", "otto")
    die("Unable to connect to SESAM");
?>
```

sesam_disconnect (unknown)

Detach from SESAM connection

```
boolean sesam_disconnect(void);
```

Returns: always TRUE.

sesam_disconnect() closes the logical link to a SESAM database (without actually disconnecting and unloading the driver).

Note that this isn't usually necessary, as the open connection is automatically closed at the end of the script's execution. Uncommitted data will be discarded, because an implicit **sesam_rollback()** is executed.

sesam_disconnect() will not close the persistent link, it will only invalidate the currently defined "*catalog*", "*schema*" and "*user*" triple, so that any sesam function called after **sesam_disconnect()** will fail.

See also: **sesam_connect()**.

Example 1. Closing a SESAM connection

```
if (sesam_connect ("mycatalog", "myschema", "otto")) {
    ... some queries and stuff ...
    sesam_disconnect();
}
```

sesam_settransaction (unknown)

Set SESAM transaction parameters

```
boolean sesam_settransaction (int isolation_level, int read_only)
```

Returns: TRUE if the values are valid, and the `settransaction()` operation was successful, FALSE otherwise.

`sesam_settransaction()` overrides the default values for the "isolation level" and "read-only" transaction parameters (which are set in the SESAM configuration file), in order to optimize subsequent queries and guarantee database consistency. The overridden values are used for the next transaction only.

`sesam_settransaction()` can only be called before starting a transaction, not after the transaction has been started already.

To simplify the use in php scripts, the following constants have been predefined in php (see SESAM handbook for detailed explanation of the semantics):

Table 1. Valid values for "*Isolation_Level*" parameter

Value	Constant	Meaning
1	SESAM_TXISOL_READ_UNCOMMITTED	Read Uncommitted
2	SESAM_TXISOL_READ_COMMITTED	Read Committed
3	SESAM_TXISOL_REPEATABLE_READ	Repeatable Read
4	SESAM_TXISOL_SERIALIZABLE	Serializable

Table 2. Valid values for "*Read_Only*" parameter

Value	Constant	Meaning
0	SESAM_TXREAD_READWRITE	Read/Write
1	SESAM_TXREAD_READONLY	Read-Only

The values set by `sesam_settransaction()` will override the default setting specified in the [SESAM configuration file](#).

Example 1. Setting SESAM transaction parameters

```
<?php
sesam_settransaction (SESAM_TXISOL_REPEATABLE_READ,
                      SESAM_TXREAD_READONLY);
?>
```

sesam_commit (unknown)

Commit pending updates to the SESAM database

```
boolean sesam_commit(void);
```

Returns: TRUE on success, FALSE on errors

sesam_commit() commits any pending updates to the database.

Note that there is no "auto-commit" feature as in other databases, as it could lead to accidental data loss. Uncommitted data at the end of the current script (or when calling **sesam_disconnect()**) will be discarded by an implied **sesam_rollback()** call.

See also: **sesam_rollback()**.

Example 1. Committing an update to the SESAM database

```
<?php
if (sesam_connect ("mycatalog", "myschema", "otto")) {
    if (!sesam_execimm ("INSERT INTO mytable VALUES (*, 'Small Test', <0, 8, 15>)"))
        die("insert failed");
    if (!sesam_commit())
        die("commit failed");
}
?>
```

sesam_rollback (unknown)

Discard any pending updates to the SESAM database

```
boolean sesam_rollback(void);
```

Returns: TRUE on success, FALSE on errors

sesam_rollback() discards any pending updates to the database. Also affected are result cursors and result descriptors.

At the end of each script, and as part of the **sesam_disconnect()** function, an implied **sesam_rollback()** is executed, discarding any pending changes to the database.

See also: **sesam_commit()**.

Example 1. Discarding an update to the SESAM database

```
<?php
if (sesam_connect ("mycatalog", "myschema", "otto")) {
    if (sesam_execimm ("INSERT INTO mytable VALUES (*, 'Small Test', <0, 8, 15>)")
        && sesam_execimm ("INSERT INTO othertable VALUES (*, 'Another Test', 1)"))
        sesam_commit();
    else
        sesam_rollback();
}
?>
```

sesam_execimm (unknown)

Execute an "immediate" SQL-statement

```
string sesam_execimm (string query)
```

Returns: A SESAM "result identifier" on success, or FALSE on error.

sesam_execimm() executes an "immediate" statement (i.e., a statement like UPDATE, INSERT or DELETE which returns no result, and has no INPUT or OUTPUT variables). "select type" queries can not be used with **sesam_execimm()**. Sets the *affected_rows* value for retrieval by the **sesam_affected_rows()** function.

Note that **sesam_query()** can handle both "immediate" and "select-type" queries. Use **sesam_execimm()** only if you know beforehand what type of statement will be executed. An attempt to use SELECT type queries with **sesam_execimm()** will return \$err["sqlstate"] == "42SBW".

The returned "result identifier" can not be used for retrieving anything but the **sesam_affected_rows()**; it is only returned for symmetry with the **sesam_query()** function.

```
$stmt = "INSERT INTO mytable VALUES ('one', 'two')";
$result = sesam_execimm ($stmt);
$err = sesam_diagnostic();
print ("sqlstate = ".$err["sqlstate"]."\n".
      "Affected rows = ".$err["rowcount"]." == ".
      sesam_affected_rows($result)."\n");
```

See also: **sesam_query()** and **sesam_affected_rows()**.

sesam_query (unknown)

Perform a SESAM SQL query and prepare the result

```
string sesam_query (string query [, boolean scrollable])
```

Returns: A SESAM "result identifier" on success, or FALSE on error.

A "result_id" resource is used by other functions to retrieve the query results.

sesam_query() sends a query to the currently active database on the server. It can execute both "immediate" SQL statements and "select type" queries. If an "immediate" statement is executed, then no cursor is allocated, and any subsequent **sesam_fetch_row()** or **sesam_fetch_result()** call will return an empty result (zero columns, indicating end-of-result). For "select type" statements, a result descriptor and a (scrollable or sequential, depending on the optional boolean *scrollable* parameter) cursor will be allocated. If *scrollable* is omitted, the cursor will be sequential.

When using "scrollable" cursors, the cursor can be freely positioned on the result set. For each "scrollable" query, there are global default values for the scrolling type (initialized to: SESAM_SEEK_NEXT) and the scrolling offset which can either be set once by **sesam_seek_row()** or each time when fetching a row using **sesam_fetch_row()**.

For "immediate" statements, the number of affected rows is saved for retrieval by the **sesam_affected_rows()** function.

See also: **sesam_fetch_row()** and **sesam_fetch_result()**.

Example 1. Show all rows of the "phone" table as a html table

```
<?php
if (!sesam_connect ("phonedb", "demo", "otto"))
    die ("cannot connect");
$result = sesam_query ("select * from phone");
if (!$result) {
    $err = sesam_diagnostic();
    die ($err["errmsg"]);
}
echo "<TABLE BORDER>\n";
// Add title header with column names above the result:
```

```

if ($cols = sesam_field_array ($result)) {
    echo " <TR><TH COLSPAN=". $cols["count"].">Result:</TH></TR>\n";
    echo " <TR>\n";
    for ($col = 0; $col < $cols["count"]; ++$col) {
        $colattr = $cols[$col];
        /* Span the table head over SESAM's "Multiple Fields": */
        if ($colattr["count"] > 1) {
            echo " <TH COLSPAN=". $colattr["count"].">". $colattr["name"] .
                "(1..". $colattr["count"].")</TH>\n";
            $col += $colattr["count"] - 1;
        } else
            echo " <TH>". $colattr["name"]. "</TH>\n";
    }
    echo " </TR>\n";
}

do {
    // Fetch the result in chunks of 100 rows max.
    $ok = sesam_fetch_result ($result, 100);
    for ($row=0; $row < $ok["rows"]; ++$row) {
        echo " <TR>\n";
        for ($col = 0; $col < $ok["cols"]; ++$col) {
            if (isset($ok[$col][$row]))
                echo " <TD>". $ok[$col][$row]. "</TD>\n";
            } else {
                echo " <TD>-empty-</TD>\n";
            }
        }
        echo " </TR>\n";
    }
}
while ($ok["truncated"]) { // while there may be more data
    echo "</TABLE>\n";
}
// free result id
sesam_free_result($result);
?>

```

sesam_num_fields (unknown)

Return the number of fields/columns in a result set

```
int sesam_num_fields (string result_id)
```

After calling **sesam_query()** with a "select type" query, this function gives you the number of columns in the result. Returns an integer describing the total number of columns (aka. fields) in the current *result_id* result set or FALSE on error.

For "immediate" statements, the value zero is returned. The SESAM "multiple field" columns count as their respective dimension, i.e., a three-column "multiple field" counts as three columns.

See also: **sesam_query()** and **sesam_field_array()** for a way to distinguish between "multiple field" columns and regular columns.

sesam_field_name (unknown)

Return one column name of the result set

```
int sesam_field_name (string result_id, int index)
```

Returns the name of a field (i.e., the column name) in the result set, or FALSE on error.

For "immediate" queries, or for dynamic columns, an empty string is returned.

Note: The column index is zero-based, not one-based as in SESAM.

See also: **sesam_field_array()**. It provides an easier interface to access the column names and types, and allows for detection of "multiple fields".

sesam_diagnostic (unknown)

Return status information for last SESAM call

```
array sesam_diagnostic(void);
```

Returns an associative array of status and return codes for the last SQL query/statement/command. Elements of the array are:

Table 1. Status information returned by sesam_diagnostic()

Element	Contents
\$array["sqlstate"]	5 digit SQL return code (see the SESAM manual for the description of the possible values of SQLSTATE)
\$array["rowcount"]	number of affected rows in last update/insert/delete (set after "immediate" statements only)
\$array["errmsg"]	"human readable" error message string (set after errors only)
\$array["errcol"]	error column number of previous error (0-based; or -1 if undefined. Set after errors only)
\$array["errlin"]	error line number of previous error (0-based; or -1 if undefined. Set after errors only)

In the following example, a syntax error (E SEW42AE ILLEGAL CHARACTER) is displayed by including the offending SQL statement and pointing to the error location:

Example 1. Displaying SESAM error messages with error position

```
<?php
// Function which prints a formatted error message,
// displaying a pointer to the syntax error in the
// SQL statement
function PrintReturncode ($exec_str) {
    $err = Sesam_Diagnostic();
    $colspan=4; // 4 cols for: sqlstate, errlin, errcol, rowcount
    if ($err["errlin"] == -1)
        -$colspan;
```

```

if ($err["errcol"] == -1)
    -$colspan;
if ($err["rowcount"] == 0)
    -$colspan;
echo "<TABLE BORDER>\n";
echo "<TR><TH COLSPAN=". $colspan ."><FONT COLOR=red>ERROR:</FONT> " .
htmlspecialchars($err["errormsg"]). "</TH></TR>\n";
if ($err["errcol"] >= 0) {
    echo "<TR><TD COLSPAN=". $colspan ."><PRE>\n";
    $errstmt = $exec_str."\n";
    for ($lin=0; $errstmt != ""; ++$lin) {
        if ($lin != $err["errlin"]) { // $lin is less or greater than errlin
            if (!( $i = strchr ($errstmt, "\n")))
                $i = "";
            $line = substr ($errstmt, 0, strlen($errstmt)-strlen($i)+1);
            $errstmt = substr($i, 1);
            if ($line != "\n")
                print htmlspecialchars ($line);
        } else {
            if (!( $i = strchr ($errstmt, "\n")))
                $i = "";
            $line = substr ($errstmt, 0, strlen ($errstmt)-strlen($i)+1);
            $errstmt = substr($i, 1);
            for ($col=0; $col < $err["errcol"]; ++$col)
                echo (substr($line, $col, 1) == "\t") ? "\t" : ".";
            echo "<FONT COLOR=RED><BLINK>\\"</BLINK></FONT>\n";
            print "<FONT COLOR=\#880000\>" . htmlspecialchars($line). "</FONT>" ;
            for ($col=0; $col < $err["errcol"]; ++$col)
                echo (substr ($line, $col, 1) == "\t") ? "\t" : ".";
            echo "<FONT COLOR=RED><BLINK></BLINK></FONT>\n";
        }
    }
    echo "</PRE></TD></TR>\n";
}
echo "<TR>\n";
echo " <TD>sqlstate=" . $err["sqlstate"] . "</TD>\n";
if ($err["errlin"] != -1)
    echo " <TD>errlin=" . $err["errlin"] . "</TD>\n";
if ($err["errcol"] != -1)
    echo " <TD>errcol=" . $err["errcol"] . "</TD>\n";
if ($err["rowcount"] != 0)
    echo " <TD>rowcount=" . $err["rowcount"] . "</TD>\n";
echo "</TR>\n";
echo "</TABLE>\n";
}

if (!sesam_connect ("mycatalog", "phoneno", "otto"))
    die ("cannot connect");

$stmt = "SELECT * FROM phone\n".
    " WHERE@ LASTNAME='KRAEMER'\n".
    " ORDER BY FIRSTNAME";
if (!$result = sesam_query ($stmt))
    PrintReturncode ($stmt);
?>

```

See also: **sesam_errormsg()** for simple access to the error string only

sesam_fetch_result (unknown)

Return all or part of a query result

```
mixed sesam_fetch_result (string result_id [, int max_rows])
```

Returns a mixed array with the query result entries, optionally limited to a maximum of *max_rows* rows. Note that both row and column indexes are zero-based.

Table 1. Mixed result set returned by sesam_fetch_result()

Array Element	Contents
int \$arr["count"]	number of columns in result set (or zero if this was an "immediate" query)
int \$arr["rows"]	number of rows in result set (between zero and <i>max_rows</i>)
boolean \$arr["truncated"]	TRUE if the number of rows was at least <i>max_rows</i> , FALSE otherwise. Note that even when this is TRUE, the next sesam_fetch_result() call may return zero rows because there are no more result entries.
mixed \$arr[col][row]	result data for all the fields at row(<i>row</i>) and column(<i>col</i>), (where the integer index <i>row</i> is between 0 and \$arr["rows"]-1, and <i>col</i> is between 0 and \$arr["count"]-1). Fields may be empty, so you must check for the existence of a field by using the php isset() function. The type of the returned fields depend on the respective SQL type declared for its column (see SESAM overview for the conversions applied). SESAM "multiple fields" are "inlined" and treated like a sequence of columns.

Note that the amount of memory used up by a large query may be gigantic. Use the *max_rows* parameter to limit the maximum number of rows returned, unless you are absolutely sure that your result will not use up all available memory.

See also: **sesam_fetch_row()**, and **sesam_field_array()** to check for "multiple fields". See the description of the **sesam_query()** function for a complete example using **sesam_fetch_result()**.

sesam_affected_rows (unknown)

Get number of rows affected by an immediate query

```
int sesam_affected_rows (string result_id)
```

result_id is a valid result id returned by **sesam_query()**.

Returns the number of rows affected by a query associated with *result_id*.

The **sesam_affected_rows()** function can only return useful values when used in combination with "immediate" SQL statements (updating operations like `INSERT`, `UPDATE` and `DELETE`) because SESAM does not deliver any "affected rows" information for "select type" queries. The number returned is the number of affected rows.

See also: **sesam_query()** and **sesam_execimm()**

```
$result = sesam_execimm ("DELETE FROM PHONE WHERE LASTNAME = '".strtoupper ($name)."'");
if (!$result) {
    ... error ...
}
print sesam_affected_rows ($result).
    " entries with last name ".$name." deleted.\n"
```

sesam_errormsg (unknown)

Returns error message of last SESAM call

```
string sesam_errormsg(void);
```

Returns the SESAM error message associated with the most recent SESAM error.

```
if (!sesam_execimm ($stmt))
    printf ("%s<br>\n", sesam_errormsg());
```

See also: **sesam_diagnostic()** for the full set of SESAM SQL status information

sesam_field_array (unknown)

Return meta information about individual columns in a result

```
array sesam_field_array (string result_id)
```

result_id is a valid result id returned by **sesam_query()**.

Returns a mixed associative/indexed array with meta information (column name, type, precision, ...) about individual columns of the result after the query associated with *result_id*.

Table 1. Mixed result set returned by sesam_field_array()

Array Element	Contents
int \$arr["count"]	Total number of columns in result set (or zero if this was an "immediate" query). SESAM "multiple fields" are "inlined" and treated like the respective number of columns.
string \$arr[col]["name"]	column name for column(col), where col is between 0 and \$arr["count"]-1. The returned value can be the empty string (for dynamically computed columns). SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same column name.

Array Element	Contents
string \$arr[col]["count"]	The "count" attribute describes the repetition factor when the column has been declared as a "multiple field". Usually, the "count" attribute is 1. The first column of a "multiple field" column however contains the number of repetitions (the second and following column of the "multiple field" contain a "count" attribute of 1). This can be used to detect "multiple fields" in the result set. See the example shown in the sesam_query() description for a sample use of the "count" attribute .
string \$arr[col]["type"]	php variable type of the data for column(col), where col is between 0 and \$arr["count"]-1. The returned value can be one of <ul style="list-style-type: none"> • "integer" • "double" • "string" depending on the SQL type of the result. SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same php type.
string \$arr[col]["sqltype"]	SQL variable type of the column data for column(col), where col is between 0 and \$arr["count"]-1. The returned value can be one of <ul style="list-style-type: none"> • "CHARACTER" <ul style="list-style-type: none"> • "VARCHAR" • "NUMERIC" • "DECIMAL" • "INTEGER" • "SMALLINT" • "FLOAT" • "REAL" • "DOUBLE" • "DATE" • "TIME" • "TIMESTAMP" describing the SQL type of the result. SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same SQL type.

Array Element	Contents
string \$arr[col]["length"]	The SQL "length" attribute of the SQL variable in column(col), where col is between 0 and \$arr["count"]-1. The "length" attribute is used with "CHARACTER" and "VARCHAR" SQL types to specify the (maximum) length of the string variable. SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same length attribute.
string \$arr[col]["precision"]	The "precision" attribute of the SQL variable in column(col), where col is between 0 and \$arr["count"]-1. The "precision" attribute is used with numeric and time data types. SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same precision attribute.
string \$arr[col]["scale"]	The "scale" attribute of the SQL variable in column(col), where col is between 0 and \$arr["count"]-1. The "scale" attribute is used with numeric data types. SESAM "multiple fields" are "inlined" and treated like the respective number of columns, each with the same scale attribute.

See the **sesam_query()** function for an example of the **sesam_field_array()** use.

sesam_fetch_row (unknown)

Fetch one row as an array

```
array sesam_fetch_row (string result_id [, int whence [, int offset]])
```

Returns an array that corresponds to the fetched row, or FALSE if there are no more rows.

The number of columns in the result set is returned in an associative array element \$array["count"]. Because some of the result columns may be empty, the **count()** function can not be used on the result row returned by **sesam_fetch_row()**.

result_id is a valid result id returned by **sesam_query()** (select type queries only!).

whence is an optional parameter for a fetch operation on "scrollable" cursors, which can be set to the following predefined constants:

Table 1. Valid values for "whence" parameter

Value	Constant	Meaning
0	SESAM_SEEK_NEXT	read sequentially (after fetch, the internal default is set to SESAM_SEEK_NEXT)
1	SESAM_SEEK_PRIOR	read sequentially backwards (after fetch, the internal default is set to SESAM_SEEK_PRIOR)

Value	Constant	Meaning
2	SESAM_SEEK_FIRST	rewind to first row (after fetch, the default is set to SESAM_SEEK_NEXT)
3	SESAM_SEEK_LAST	seek to last row (after fetch, the default is set to SESAM_SEEK_PRIOR)
4	SESAM_SEEK_ABSOLUTE	seek to absolute row number given as <i>offset</i> (<i>Zero-based. After fetch, the internal default is set to SESAM_SEEK_ABSOLUTE, and the internal offset value is auto-incremented</i>)
5	SESAM_SEEK_RELATIVE	seek relative to current scroll position, where <i>offset</i> can be a positive or negative offset value.

This parameter is only valid for "scrollable" cursors.

When using "scrollable" cursors, the cursor can be freely positioned on the result set. If the *whence* parameter is omitted, the global default values for the scrolling type (initialized to: SESAM_SEEK_NEXT, and settable by **sesam_seek_row()**) are used. If *whence* is supplied, its value replaces the global default.

offset is an optional parameter which is only evaluated (and required) if *whence* is either SESAM_SEEK_RELATIVE or SESAM_SEEK_ABSOLUTE. This parameter is only valid for "scrollable" cursors.

sesam_fetch_row() fetches one row of data from the result associated with the specified result identifier. The row is returned as an array (indexed by values between 0 and \$array["count"]-1). Fields may be empty, so you must check for the existence of a field by using the php **isset()** function. The type of the returned fields depend on the respective SQL type declared for its column (see [SESAM overview](#) for the conversions applied). SESAM "multiple fields" are "inlined" and treated like a sequence of columns.

Subsequent calls to **sesam_fetch_row()** would return the next (or prior, or n'th next/prior, depending on the scroll attributes) row in the result set, or FALSE if there are no more rows.

Example 1. SESAM fetch rows

```
<?php
$result = sesam_query ("SELECT * FROM phone\n".
                      " WHERE LASTNAME='".strtoupper($name)."' \n".
                      " ORDER BY FIRSTNAME", 1);
if (!$result) {
    ... error ...
}
// print the table in backward order
print "<TABLE BORDER>\n";
$row = sesam_fetch_row ($result, SESAM_SEEK_LAST);
while (is_array ($row)) {
    print " <TR>\n";
    for ($col = 0; $col < $row["count"]; ++$col) {
        print "   <TD>".htmlspecialchars ($row[$col])."</TD>\n";
    }
    print " </TR>\n";
    // use implied SESAM_SEEK_PRIOR
    $row = sesam_fetch_row ($result);
}
print "</TABLE>\n";
sesam_free_result ($result);
?>
```

See also: **sesam_fetch_array()** which returns an associative array, and **sesam_fetch_result()** which returns many rows per invocation.

sesam_fetch_array (unknown)

Fetch one row as an associative array

```
array sesam_fetch_array (string result_id [, int whence [, int offset]])
```

Returns an array that corresponds to the fetched row, or FALSE if there are no more rows.

sesam_fetch_array() is an alternative version of **sesam_fetch_row()**. Instead of storing the data in the numeric indices of the result array, it stores the data in associative indices, using the field names as keys.

result_id is a valid result id returned by **sesam_query()** (select type queries only!).

For the valid values of the optional *whence* and *offset* parameters, see the **sesam_fetch_row()** function for details.

sesam_fetch_array() fetches one row of data from the result associated with the specified result identifier. The row is returned as an associative array. Each result column is stored with an associative index equal to its column (aka. field) name. The column names are converted to lower case.

Columns without a field name (e.g., results of arithmetic operations) and empty fields are not stored in the array. Also, if two or more columns of the result have the same column names, the later column will take precedence. In this situation, either call **sesam_fetch_row()** or make an alias for the column.

```
SELECT TBL1.COL AS FOO, TBL2.COL AS BAR FROM TBL1, TBL2
```

A special handling allows fetching "multiple field" columns (which would otherwise all have the same column names). For each column of a "multiple field", the index name is constructed by appending the string "(n)" where n is the sub-index of the multiple field column, ranging from 1 to its declared repetition factor. The indices are NOT zero based, in order to match the nomenclature used in the respective query syntax. For a column declared as:

```
CREATE TABLE ... ( ... MULTI(3) INT )
```

the associative indices used for the individual "multiple field" columns would be "multi(1)", "multi(2)", and "multi(3)" respectively.

Subsequent calls to **sesam_fetch_array()** would return the next (or prior, or n'th next/prior, depending on the scroll attributes) row in the result set, or FALSE if there are no more rows.

Example 1. SESAM fetch array

```
<?php
$result = sesam_query ("SELECT * FROM phone\n".
                      " WHERE LASTNAME='".strtoupper($name)."'\\n".
                      " ORDER BY FIRSTNAME", 1);
if (!$result) {
    ... error ...
}
// print the table:
print "<TABLE BORDER>\\n";
while (($row = sesam_fetch_array ($result)) && count ($row) > 0) {
    print " <TR>\\n";
    print " <TD>".htmlspecialchars ($row["firstname"])."</TD>\\n";
    print " <TD>".htmlspecialchars ($row["lastname"])."</TD>\\n";
    print " <TD>".htmlspecialchars ($row["phoneno"])."</TD>\\n";
    print " </TR>\\n";
```

```

}
print "</TABLE>\n";
sesam_free_result ($result);
?>

```

See also: **sesam_fetch_row()** which returns an indexed array.

sesam_seek_row (unknown)

Set scrollable cursor mode for subsequent fetches

```
boolean sesam_seek_row (string result_id, int whence [, int offset])
```

result_id is a valid result id (select type queries only, and only if a "scrollable" cursor was requested when calling **sesam_query()**).

whence sets the global default value for the scrolling type, it specifies the scroll type to use in subsequent fetch operations on "scrollable" cursors, which can be set to the following predefined constants:

Table 1. Valid values for "whence" parameter

Value	Constant	Meaning
0	SESAM_SEEK_NEXT	read sequentially
1	SESAM_SEEK_PRIOR	read sequentially backwards
2	SESAM_SEEK_FIRST	fetch first row (after fetch, the default is set to SESAM_SEEK_NEXT)
3	SESAM_SEEK_LAST	fetch last row (after fetch, the default is set to SESAM_SEEK_PRIOR)
4	SESAM_SEEK_ABSOLUTE	fetch absolute row number given as <i>offset</i> (<i>Zero-based. After fetch, the default is set to SESAM_SEEK_ABSOLUTE, and the offset value is auto-incremented</i>)
5	SESAM_SEEK_RELATIVE	fetch relative to current scroll position, where <i>offset</i> can be a positive or negative offset value (<i>this also sets the default "offset" value for subsequent fetches</i>).

offset is an optional parameter which is only evaluated (and required) if *whence* is either SESAM_SEEK_RELATIVE or SESAM_SEEK_ABSOLUTE.

sesam_free_result (unknown)

Releases resources for the query

```
int sesam_free_result (string result_id)
```

Releases resources for the query associated with *result_id*. Returns FALSE on error.

LXV. Session handling functions

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

If you are familiar with the session management of PHPLIB, you will notice that some concepts are similar to PHP's session support.

A visitor accessing your web site is assigned an unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.

The session support allows you to register arbitrary numbers of variables to be preserved across requests. When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through **session_start()** or implicitly through **session_register()**) whether a specific session id has been sent with the request. If this is the case, the prior saved environment is recreated.

All registered variables are serialized after the request finishes. Registered variables which are undefined are marked as being not defined. On subsequent accesses, these are not defined by the session module unless the user defines them later.

The **track_vars** and **register_globals** configuration settings influence how the session variables get stored and restored.

Note: As of PHP 4.0.3, **track_vars** is always turned on.

If **track_vars** is enabled and **register_globals** is disabled, only members of the global associative array **\$HTTP_SESSION_VARS** can be registered as session variables. The restored session variables will only be available in the array **\$HTTP_SESSION_VARS**.

Example 1. Registering a variable with **track_vars** enabled

```
<?php
session_register( "count" );
$HTTP_SESSION_VARS[ "count" ]++;
?>
```

If **register_globals** is enabled, then all global variables can be registered as session variables and the session variables will be restored to corresponding global variables.

Example 2. Registering a variable with **register_globals** enabled

```
<?php
session_register( "count" );
$count++;
?>
```

If both **track_vars** and **register_globals** are enabled, then the global variables and the **\$HTTP_SESSION_VARS** entries will reference the same value.

There are two methods to propagate a session id:

- Cookies
- URL parameter

The session module supports both methods. Cookies are optimal, but since they are not reliable (clients are not bound to

accept them), we cannot rely on them. The second method embeds the session id directly into URLs.

PHP is capable of doing this transparently when compiled with `-enable-trans-sid`. If you enable this option, relative URIs will be changed to contain the session id automatically. Alternatively, you can use the constant `SID` which is defined, if the client did not send the appropriate cookie. `SID` is either of the form `session_name=session_id` or is an empty string.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

Example 3. Counting the number of hits of a single user

```
<?php
session_register ("count");
$count++;
?>

Hello visitor, you have seen this page <?php echo $count; ?> times.<p>

<php?
# the <?=SID?> is necessary to preserve the session id
# in the case that the user has disabled cookies
?>

To continue, <A HREF="nextpage.php?<?=SID?>">click here</A>
```

The `<?=SID?>` is not necessary, if `-enable-trans-sid` was used to compile PHP.

To implement database storage, or any other storage method, you will need to use `session_set_save_handler()` to create a set of user-level storage functions.

The session management system supports a number of configuration options which you can place in your `php.ini` file. We will give a short overview.

- `session.save_handler` defines the name of the handler which is used for storing and retrieving data associated with a session. Defaults to `files`.
- `session.save_path` defines the argument which is passed to the save handler. If you choose the default files handler, this is the path where the files are created. Defaults to `/tmp`.

Warning

If you leave this set to a world-readable directory, such as `/tmp` (the default), other users on the server may be able to hijack sessions by getting the list of files in that directory.

- `session.name` specifies the name of the session which is used as cookie name. It should only contain alphanumeric characters. Defaults to `PHPSESSID`.
- `session.auto_start` specifies whether the session module starts a session automatically on request startup. Defaults to 0 (disabled).
- `session.cookie_lifetime` specifies the lifetime of the cookie in seconds which is sent to the browser. The value 0 means "until the browser is closed." Defaults to 0.
- `session.serialize_handler` defines the name of the handler which is used to serialize/deserialize data. Currently, a PHP internal format (name `php`) and WDDX is supported (name `wddx`). WDDX is only available, if PHP is compiled

with [WDDX support](#). Defaults to `php`.

- `session.gc_probability` specifies the probability that the gc (garbage collection) routine is started on each request in percent. Defaults to 1.
- `session.gc_maxlifetime` specifies the number of seconds after which data will be seen as 'garbage' and cleaned up.
- `session.referer_check` contains the substring you want to check each HTTP Referer for. If the Referer was sent by the client and the substring was not found, the embedded session id will be marked as invalid. Defaults to the empty string.
- `session.entropy_file` gives a path to an external resource (file) which will be used as an additional entropy source in the session id creation process. Examples are `/dev/random` or `/dev/urandom` which are available on many Unix systems.
- `session.entropy_length` specifies the number of bytes which will be read from the file specified above. Defaults to 0 (disabled).
- `session.use_cookies` specifies whether the module will use cookies to store the session id on the client side. Defaults to 1 (enabled).
- `session.cookie_path` specifies path to set in `session_cookie`. Defaults to `/`.
- `session.cookie_domain` specifies domain to set in `session_cookie`. Default is none at all.
- `session.cache_limiter` specifies cache control method to use for session pages (nocache/private/public). Defaults to `nocache`.
- `session.cache_expire` specifies time-to-live for cached session pages in minutes, this has no effect for `nocache` limiter. Defaults to 180.

Note: Session handling was added in PHP 4.0.

session_start (PHP 4)

Initialize session data

```
bool session_start(void);
```

session_start() creates a session (or resumes the current one based on the session id being passed via a GET variable or a cookie).

This function always returns true.

Note: This function was added in PHP 4.0.

session_destroy (PHP 4)

Destroys all data registered to a session

```
bool session_destroy(void);
```

session_destroy() destroys all of the data associated with the current session.

This function returns true on success and false on failure to destroy the session data.

session_name (PHP 4)

Get and/or set the current session name

```
string session_name ([string name])
```

session_name() returns the name of the current session. If *name* is specified, the name of the current session is changed to its value.

The session name references the session id in cookies and URLs. It should contain only alphanumeric characters; it should be short and descriptive (i.e. for users with enabled cookie warnings). The session name is reset to the default value stored in `session.name` at request startup time. Thus, you need to call **session_name()** for every request (and before **session_start()** or **session_register()** are called).

Example 1. **session_name()** examples

```
<?php

# set the session name to WebsiteID

$previous_name = session_name ("WebsiteID");

echo "The previous session name was $previous_name<p>";
?>
```

Note: This function was added in PHP 4.0.

session_module_name (PHP 4)

Get and/or set the current session module

```
string session_module_name ([string module])
```

session_module_name() returns the name of the current session module. If *module* is specified, that module will be used instead.

Note: This function was added in PHP 4.0.

session_save_path (PHP 4)

Get and/or set the current session save path

```
string session_save_path ([string path])
```

session_save_path() returns the path of the current directory used to save session data. If *path* is specified, the path to which data is saved will be changed.

Note: On some operating systems, you may want to specify a path on a filesystem that handles lots of small files efficiently. For example, on Linux, reiserfs may provide better performance than ext2fs.

Note: This function was added in PHP 4.0.

session_id (PHP 4)

Get and/or set the current session id

```
string session_id ([string id])
```

session_id() returns the session id for the current session. If *id* is specified, it will replace the current session id.

The constant SID can also be used to retrieve the current name and session id as a string suitable for adding to URLs.

session_register (PHP 4)

Register one or more variables with the current session

```
bool session_register (mixed name [, mixed ...])
```

session_register() variable number of arguments, any of which can be either a string holding the variable name or an array consisting of such variable names or other arrays. For each encountered variable name, **session_register()** registers the global variable named by it with the current session.

This function returns true when the variable is successfully registered with the session.

Note: This function was added in PHP 4.0.

session_unregister (PHP 4)

Unregister a variable from the current session

```
bool session_unregister (string name)
```

session_unregister() unregisters (forgets) the global variable named *name* from the current session.

This function returns true when the variable is successfully unregistered from the session.

Note: This function was added in PHP 4.0.

session_unset (PHP 4 >= 4.0b4)

Free all session variables

```
void session_unset(void);
```

The **session_unset()** function free's all session variables currently registered.

session_is_registered (PHP 4)

Find out if a variable is registered in a session

```
bool session_is_registered (string name)
```

session_is_registered() returns true if there is a variable with the name *name* registered in the current session.

Note: This function was added in PHP 4.0.

session_get_cookie_params (PHP 4 >= 4.0RC2)

Get the session cookie parameters

```
array session_get_cookie_params (void);
```

The **session_get_cookie_params()** function returns an array with the current session cookie information, the array contains the following items:

- "lifetime" - The lifetime of the cookie.
- "path" - The path where information is stored.
- "domain" - The domain of the cookie.

session_set_cookie_params (PHP 4 >= 4.0b4)

Set the session cookie parameters

```
void session_set_cookie_params (int lifetime [, string path [, string domain]])
```

Set cookie parameters defined in the php.ini file. The effect of this function only lasts for the duration of the script.

session_decode (PHP 4)

Decodes session data from a string

```
bool session_decode (string data)
```

session_decode() decodes the session data in *data*, setting variables stored in the session.

Note: This function was added in PHP 4.0.

session_encode (PHP 4)

Encodes the current session data as a string

```
string session_encode(void);
```

session_encode() returns a string with the contents of the current session encoded within.

Note: This function was added in PHP 4.0.

session_set_save_handler (PHP 4 >= 4.0b4)

Sets user-level session storage functions

```
void session_set_save_handler (string open, string close, string read, string write,
string destroy, string gc)
```

session_set_save_handler() sets the user-level session storage functions which are used for storing and retrieving data associated with a session. This is most useful when a storage method other than those supplied by PHP sessions is preferred. i.e. Storing the session data in a local database.

Note: You must set the configuration option `session.save_handler` to `user` in your `php.ini` file for `session_set_save_handler()` to take effect.

Note: The "write" handler is not executed until after the output stream is closed. Thus, output from debugging statements in the "write" handler will never be seen in the browser. If debugging output is necessary, it is suggested that the debug output be written to a file instead.

The following example provides file based session storage similar to the PHP sessions default save handler *files*. This example could easily be extended to cover database storage using your favorite PHP supported database engine.

Example 1. `session_set_save_handler()` example

```
<?php

function open ($save_path, $session_name) {
    global $sess_save_path, $sess_session_name;

    $sess_save_path = $save_path;
    $sess_session_name = $session_name;
    return(true);
}

function close() {
    return(true);
}

function read ($id) {
    global $sess_save_path, $sess_session_name;

    $sess_file = "$sess_save_path/sess_$id";
    if ($fp = @fopen($sess_file, "r")) {
        $sess_data = fread($fp, filesize($sess_file));
        return($sess_data);
    } else {
        return("");
    }
}

function write ($id, $sess_data) {
    global $sess_save_path, $sess_session_name;

    $sess_file = "$sess_save_path/sess_$id";
    if ($fp = @fopen($sess_file, "w")) {
        return(fwrite($fp, $sess_data));
    } else {
        return(false);
    }
}
```

```

function destroy ($id) {
    global $sess_save_path, $sess_session_name;

    $sess_file = "$sess_save_path/sess_$id";
    return(@unlink($sess_file));
}

/*
 * WARNING - You will need to implement some
 * sort of garbage collection routine here.
 */
function gc ($maxlifetime) {
    return true;
}

session_set_save_handler ("open", "close", "read", "write", "destroy", "gc");

session_start();

// proceed to use sessions normally

?>

```

session_cache_limiter (PHP 4 >= 4.0.3)

Get and/or set the current cache limiter

```
string session_cache_limiter ([string cache_limiter])
```

session_cache_limiter() returns the name of the current cache limiter. If *cache_limiter* is specified, the name of the current cache limiter is changed to the new value.

The cache limiter controls the cache control HTTP headers sent to the client. These headers determine the rules by which the page content may be cached. Setting the cache limiter to `nocache`, for example, would disallow any client-side caching. A value of `public`, however, would permit caching. It can also be set to `private`, which is slightly more restrictive than `public`.

The cache limiter is reset to the default value stored in `session.cache_limiter` at request startup time. Thus, you need to call **session_cache_limiter()** for every request (and before **session_start()** is called).

Example 1. session_cache_limiter() examples

```
<?php

# set the cache limiter to 'private'

session_cache_limiter('private');
$cache_limiter = session_cache_limiter();

echo "The cache limiter is now set to $cache_limiter<p>";
?>
```

Note: This function was added in PHP 4.0.3.

LXVI. Shared Memory Functions

Shmop is an easy to use set of functions that allows php to read, write, create and delete UNIX shared memory segments. The functions will not work on windows, as it does not support shared memory. To use shmop you will need to compile php with the `--enable-shmop` parameter in your configure line.

Note: The functions explained in the chapter begin all with `shm_()` in PHP 4.0.3, but in PHP 4.0.4 and later versions these names are changed to begin with `shmop_()`.

Example 1. Shared Memory Operations Overview

```
<?php

// Create 100 byte shared memory block with system id if 0xff3
$shm_id = shmop_open(0xff3, "c", 0644, 100);
if(!$shm_id) {
echo "Couldn't create shared memory segment\n";
}

// Get shared memory block's size
$shm_size = shmop_size($shm_id);
echo "SHM Block Size: ".$shm_size. " has been created.\n";

// Lets write a test string into shared memory
$shm_bytes_written = shmop_write($shm_id, "my shared memory block", 0);
if($shm_bytes_written != strlen("my shared memory block")) {
echo "Couldn't write the entire length of data\n";
}

// Now lets read the string back
$my_string = shmop_read($shm_id, 0, $shm_size);
if(!$my_string) {
echo "Couldn't read from shared memory block\n";
}
echo "The data inside shared memory was: ".$my_string."\n";

//Now lets delete the block and close the shared memory segment
if(!shmop_delete($shm_id)) {
echo "Couldn't mark shared memory block for deletion.";
}
shmop_close($shm_id);

?>
```


shmop_open (unknown)

Create or open shared memory block

```
int shmop_open (int key, string flags, int mode, int size)
```

shmop_open() can create or open a shared memory block.

shmop_open() takes 4 parameters: key, which is the system's id for the shared memory block, this parameter can be passed as a decimal or hex. The second parameter are the flags that you can use:

- "a" for access (sets IPC_EXCL) use this flag when you need to open an existing shared memory segment
- "c" for create (sets IPC_CREATE) use this flag when you need to create a new shared memory segment.

The third parameter is the mode, which are the permissions that you wish to assign to your memory segment, those are the same as permission for a file. Permissions need to be passed in octal form ex. 0644. The last parameter is size of the shared memory block you wish to create in bytes.

Note: Note: the 3rd and 4th should be entered as 0 if you are opening an existing memory segment. On success **shmop_open()** will return an id that you can use to access the shared memory segment you've created.

Example 1. Create a new shared memory block

```
<?php
$shm_id = shmop_open(0x0fff, "c", 0644, 100);
?>
```

This example opened a shared memory block with a system id of 0x0fff.

shmop_read (unknown)

Read data from shared memory block

```
string shmop_read (int shmid, int start, int count)
```

shmop_read() will read a string from shared memory block.

shmop_read() takes 3 parameters: shmid, which is the shared memory block identifier created by **shmop_open()**, offset from which to start reading and count on the number of bytes to read.

Example 1. Reading shared memory block

```
<?php
$shm_data = shmop_read($shm_id, 0, 50);
?>
```

This example will read 50 bytes from shared memory block and place the data inside \$shm_data.

shmop_write (unknown)

Write data into shared memory block

```
int shmop_write (int shmid, string data, int offset)
```

shmop_write() will write a string into shared memory block.

shmop_write() takes 3 parameters: shmid, which is the shared memory block identifier created by **shmop_open()**, data, a string that you want to write into shared memory block and offset, which specifies where to start writing data inside the shared memory segment.

Example 1. Writing to shared memory block

```
<?php
$shm_bytes_written = shmop_write($shm_id, $my_string, 0);
?>
```

This example will write data inside \$my_string into shared memory block, \$shm_bytes_written will contain the number of bytes written.

shmop_size (unknown)

Get size of shared memory block

```
int shmop_size (int shmid)
```

shmop_size() is used to get the size, in bytes of the shared memory block.

shmop_size() takes the shmid, which is the shared memory block identifier created by **shmop_open()**, the function will return and int, which represents the number of bytes the shared memory block occupies.

Example 1. Getting the size of the shared memory block

```
<?php
$shm_size = shmop_size($shm_id);
?>
```

This example will put the size of shared memory block identified by \$shm_id into \$shm_size.

shmop_delete (unknown)

Delete shared memory block

```
int shmop_delete (int shmid)
```

shmop_delete() is used to delete a shared memory block.

shmop_delete() takes the shmid, which is the shared memory block identifier created by **shmop_open()**. On success 1 is returned, on failure 0 is returned.

Example 1. Deleting shared memory block

```
<?php  
shmop_delete($shm_id);  
?>
```

This example will delete shared memory block identified by \$shm_id.

shmop_close (unknown)

Close shared memory block

```
int shmop_close (int shmid)
```

shmop_close() is used to close a shared memory block.

shmop_close() takes the shmid, which is the shared memory block identifier created by **shmop_open()**.

Example 1. Closing shared memory block

```
<?php  
shmop_close($shm_id);  
?>
```

This example will close shared memory block identified by \$shm_id.

LXVII. Shockwave Flash functions

PHP offers the ability to create Shockwave Flash files via Paul Haeberli's libswf module. You can download libswf at <http://reality.sgi.com/grafica/flash/>. Once you have libswf all you need to do is to configure `-with-swf[=DIR]` where DIR is a location containing the directories include and lib. The include directory has to contain the swf.h file and the lib directory has to contain the libswf.a file. If you unpack the libswf distribution the two files will be in one directory. Consequently you will have to copy the files to the proper location manually.

Once you've successfully installed PHP with Shockwave Flash support you can then go about creating Shockwave files from PHP. You would be surprised at what you can do, take the following code:

Example 1. SWF example

```
<?php
swf_openfile ("test.swf", 256, 256, 30, 1, 1, 1);
swf_ortho2 (-100, 100, -100, 100);
swf_defineline (1, -70, 0, 70, 0, .2);
swf_definerect (4, 60, -10, 70, 0, 0);
swf_definerect (5, -60, 0, -70, 10, 0);
swf_addcolor (0, 0, 0, 0);

swf_definefont (10, "Mod");
swf_fontsize (5);
swf_fontslant (10);
swf_definetext (11, "This be Flash wit PHP!", 1);

swf_pushmatrix ();
swf_translate (-50, 80, 0);
swf_placeobject (11, 60);
swf_popmatrix ();

for ($i = 0; $i < 30; $i++) {
    $p = $i/(30-1);
    swf_pushmatrix ();
    swf_scale (1-($p*.9), 1, 1);
    swf_rotate (60*$p, 'z');
    swf_translate (20+20*$p, $p/1.5, 0);
    swf_rotate (270*$p, 'z');
    swf_addcolor ($p, 0, $p/1.2, -$p);
    swf_placeobject (1, 50);
    swf_placeobject (4, 50);
    swf_placeobject (5, 50);
    swf_popmatrix ();
    swf_showframe ();
}

for ($i = 0; $i < 30; $i++) {
    swf_removeobject (50);
    if (($i%4) == 0) {
        swf_showframe ();
    }
}

swf_startdoaction ();
swf_actionstop ();
swf_enddoaction ();

swf_closefile ();
?>
```

It will produce the animation found at the following url (<http://www.designmultimedia.com/swfphp/test.swf>).

Note: SWF support was added in PHP 4 RC2.

The libswf does not have support for Windows. The development of that library has been stopped, and the source is not available to port it to another systems.

swf_openfile (PHP 4 >= 4.0RC2)

Open a new Shockwave Flash file

```
void swf_openfile (string filename, float width, float height, float framerate, float r,  
float g, float b)
```

The **swf_openfile()** function opens a new file named *filename* with a width of *width* and a height of *height* a frame rate of *framerate* and background with a red color of *r* a green color of *g* and a blue color of *b*.

The **swf_openfile()** must be the first function you call, otherwise your script will cause a segfault. If you want to send your output to the screen make the filename: "php://stdout" (support for this is in 4.0.1 and up).

swf_closefile (PHP 4 >= 4.0RC2)

Close the current Shockwave Flash file

```
void swf_closefile ([int return_file])
```

Close a file that was opened by the **swf_openfile()** function. If the *return_file* parameter is set then the contents of the SWF file are returned from the function.

Example 1. Creating a simple flash file based on user input and outputting it and saving it in a database

```
<?php

// The $text variable is submitted by the
// user

// Global variables for database
// access (used in the swf_savedata() function)
$DBHOST = "localhost";
$DBUSER = "sterling";
$DBPASS = "secret";

swf_openfile ("php://stdout", 256, 256, 30, 1, 1, 1);

swf_definefont (10, "Ligon-Bold");
    swf_fontsize (12);
    swf_fontslant (10);

swf_definetext (11, $text, 1);

swf_pushmatrix ();
    swf_translate (-50, 80, 0);
    swf_placeobject (11, 60);
swf_popmatrix ();

swf_showframe ();

swf_startdoaction ();
    swf_actionstop ();
swf_enddoaction ();

$data = swf_closefile (1);
```

```

$data ?
    swf_savedata ($data) :
        die ("Error could not save SWF file");

// void swf_savedata (string data)
// Save the generated file a database
// for later retrieval
function swf_savedata ($data)
{
    global $DBHOST,
           $DBUSER,
           $DBPASS;

$dh = @mysql_connect ($DBHOST, $DBUSER, $DBPASS);

if (!$dh) {
    die (sprintf ("Error [%d]: %s",
                  mysql_errno (), mysql_error ()));
}

$stmt = "INSERT INTO swf_files (file) VALUES ('$data')";

$sth = @mysql_query ($stmt, $dh);

if (!$sth) {
    die (sprintf ("Error [%d]: %s",
                  mysql_errno (), mysql_error ()));
}

@mysql_free_result ($sth);
@mysql_close ($dh);
}
>

```

swf_labelframe (PHP 4 >= 4.0RC2)

Label the current frame

```
void swf_labelframe (string name)
```

Label the current frame with the name given by the *name* parameter.

swf_showframe (PHP 4 >= 4.0RC2)

Display the current frame

```
void swf_showframe (void);
```

The swf_showframe function will output the current frame.

swf_setframe (PHP 4 >= 4.0RC2)

Switch to a specified frame

```
void swf_setframe (int framenumber)
```

The **swf_setframe()** changes the active frame to the frame specified by *framenumber*.

swf_getframe (PHP 4 >= 4.0RC2)

Get the frame number of the current frame

```
int swf_getframe (void);
```

The **swf_getframe()** function gets the number of the current frame.

swf_mulcolor (PHP 4 >= 4.0RC2)

Sets the global multiply color to the *rgba* value specified

```
void swf_mulcolor (float r, float g, float b, float a)
```

The **swf_mulcolor()** function sets the global multiply color to the *rgba* color specified. This color is then used (implicitly) by the **swf_placeobject()**, **swf_modifyobject()** and the **swf_addbuttonrecord()** functions. The color of the object will be multiplied by the *rgba* values when the object is written to the screen.

Note: The *rgba* values can be either positive or negative.

swf_addcolor (PHP 4 >= 4.0RC2)

Set the global add color to the *rgba* value specified

```
void swf_addcolor (float r, float g, float b, float a)
```

The **swf_addcolor()** function sets the global add color to the *rgba* color specified. This color is then used (implicitly) by the **swf_placeobject()**, **swf_modifyobject()** and the **swf_addbuttonrecord()** functions. The color of the object will be add by the *rgba* values when the object is written to the screen.

Note: The *rgba* values can be either positive or negative.

swf_placeobject (PHP 4 >= 4.0RC2)

Place an object onto the screen

```
void swf_placeobject (int objid, int depth)
```

Places the object specified by *objid* in the current frame at a depth of *depth*. The *objid* parameter and the *depth* must be between 1 and 65535.

This uses the current mulcolor (specified by **swf_mulcolor()**) and the current addcolor (specified by **swf_addcolor()**) to color the object and it uses the current matrix to position the object.

Note: Full RGBA colors are supported.

swf_modifyobject (PHP 4 >= 4.0RC2)

Modify an object

```
void swf_modifyobject (int depth, int how)
```

Updates the position and/or color of the object at the specified depth, *depth*. The parameter *how* determines what is updated. *how* can either be the constant MOD_MATRIX or MOD_COLOR or it can be a combination of both (MOD_MATRIX|MOD_COLOR).

MOD_COLOR uses the current mulcolor (specified by the function **swf_mulcolor()**) and addcolor (specified by the function **swf_addcolor()**) to color the object. MOD_MATRIX uses the current matrix to position the object.

swf_removeobject (PHP 4 >= 4.0RC2)

Remove an object

```
void swf_removeobject (int depth)
```

Removes the object at the depth specified by *depth*.

swf_nextid (PHP 4 >= 4.0RC2)

Returns the next free object id

```
int swf_nextid (void);
```

The **swf_nextid()** function returns the next available object id.

swf_startdoaction (PHP 4 >= 4.0RC2)

Start a description of an action list for the current frame

```
void swf_startdoaction (void);
```

The **swf_startdoaction()** function starts the description of an action list for the current frame. This must be called before actions are defined for the current frame.

swf_actiongotoframe (PHP 4 >= 4.0RC2)

Play a frame and then stop

```
void swf_actiongotoframe (int framenumber)
```

The **swf_actionGotoFrame()** function will go to the frame specified by *framenumber*, play it, and then stop.

swf_actiongeturl (PHP 4 >= 4.0RC2)

Get a URL from a Shockwave Flash movie

```
void swf_actiongeturl (string url, string target)
```

The **swf_actionGetUrl()** function gets the URL specified by the parameter *url* with the target *target*.

swf_actionnextframe (PHP 4 >= 4.0RC2)

Go foward one frame

```
void swf_actionnextframe (void);
```

Go foward one frame.

swf_actionprevframe (PHP 4 >= 4.0RC2)

Go backwards one frame

```
void swf_actionprevframe (void);
```

swf_actionplay (PHP 4 >= 4.0RC2)

Start playing the flash movie from the current frame

```
void swf_actionplay (void);
```

Start playing the flash movie from the current frame.

swf_actionstop (PHP 4 >= 4.0RC2)

Stop playing the flash movie at the current frame

```
void swf_actionstop (void);
```

Stop playing the flash movie at the current frame.

swf_actiontogglequality (PHP 4 >= 4.0RC2)

Toggle between low and high quality

```
void swf_actiontogglequality (void);
```

Toggle the flash movie between high and low quality.

swf_actionwaitforframe (PHP 4 >= 4.0RC2)

Skip actions if a frame has not been loaded

```
void swf_actionwaitForframe (int framenumber, int skipcount)
```

The **swf_actionWaitForFrame()** function will check to see if the frame, specified by the *framenumber* parameter has been loaded, if not it will skip the number of actions specified by the *skipcount* parameter. This can be useful for "Loading..." type animations.

swf_actionsettarget (PHP 4 >= 4.0RC2)

Set the context for actions

```
void swf_actionsettarget (string target)
```

The **swf_actionSetTarget()** function sets the context for all actions. You can use this to control other flash movies that are currently playing.

swf_actiongotolabel (PHP 4 >= 4.0RC2)

Display a frame with the specified label

```
void swf_actiongotolabel (string label)
```

The **swf_actionGotoLabel()** function displays the frame with the label given by the *label* parameter and then stops.

swf_enddoaction (PHP 4 >= 4.0RC2)

End the current action

```
void swf_enddoaction (void);
```

Ends the current action started by the **swf_startdoaction()** function.

swf_defineline (PHP 4 >= 4.0RC2)

Define a line

```
void swf_defineline (int objid, float x1, float y1, float x2, float y2, float width)
```

The **swf_defineline()** defines a line starting from the x coordinate given by *x1* and the y coordinate given by *y1* parameter. Up to the x coordinate given by the *x2* parameter and the y coordinate given by the *y2* parameter. It will have a width defined by the *width* parameter.

swf_definerect (PHP 4 >= 4.0RC2)

Define a rectangle

```
void swf_definerect (int objid, float x1, float y1, float x2, float y2, float width)
```

The **swf_definerect()** defines a rectangle with an upper left hand coordinate given by the x, *x1*, and the y, *y1*. And a lower right hand coordinate given by the x coordinate, *x2*, and the y coordinate, *y2* . Width of the rectangles border is given by the *width* parameter, if the width is 0.0 then the rectangle is filled.

swf_definepoly (PHP 4 >= 4.0.0)

Define a polygon

```
void swf_definepoly (int objid, array coords, int npoints, float width)
```

The **swf_definepoly()** function defines a polygon given an array of x, y coordinates (the coordinates are defined in the parameter *coords*). The parameter *npoints* is the number of overall points that are contained in the array given by *coords*. The *width* is the width of the polygon's border, if set to 0.0 the polygon is filled.

swf_startshape (PHP 4 >= 4.0RC2)

Start a complex shape

```
void swf_startshape (int objid)
```

The **swf_startshape()** function starts a complex shape, with an object id given by the *objid* parameter.

swf_shapelinesolid (PHP 4 >= 4.0RC2)

Set the current line style

```
void swf_shapelinesolid (float r, float g, float b, float a, float width)
```

The **swf_shapeLineSolid()** function sets the current line style to the color of the *rgba* parameters and width to the *width* parameter. If 0.0 is given as a width then no lines are drawn.

swf_shapefilloff (PHP 4 >= 4.0RC2)

Turns off filling

```
void swf_shapefilloff (void);
```

The **swf_shapeFillOff()** function turns off filling for the current shape.

swf_shapefillsolid (PHP 4 >= 4.0RC2)

Set the current fill style to the specified color

```
void swf_shapefillsolid (float r, float g, float b, float a)
```

The **swf_shapeFillSolid()** function sets the current fill style to solid, and then sets the fill color to the values of the *rgba* parameters.

swf_shapefillbitmapclip (PHP 4 >= 4.0RC2)

Set current fill mode to clipped bitmap

```
void swf_shapefillbitmapclip (int bitmapid)
```

Sets the fill to bitmap clipped, empty spaces will be filled by the bitmap given by the *bitmapid* parameter.

swf_shapefillbitmaptile (PHP 4 >= 4.0RC2)

Set current fill mode to tiled bitmap

```
void swf_shapefillbitmaptile (int bitmapid)
```

Sets the fill to bitmap tile, empty spaces will be filled by the bitmap given by the *bitmapid* parameter (tiled).

swf_shapemoveto (PHP 4 >= 4.0RC2)

Move the current position

```
void swf_shapemoveto (float x, float y)
```

The **swf_shapeMoveTo()** function moves the current position to the x coordinate given by the *x* parameter and the y position given by the *y* parameter.

swf_shapelineto (PHP 4 >= 4.0RC2)

Draw a line

```
void swf_shapelineto (float x, float y)
```

The **swf_shapeLineTo()** draws a line to the x,y coordinates given by the *x* parameter & the *y* parameter. The current position is then set to the x,y parameters.

swf_shapecurveto (PHP 4 >= 4.0RC2)

Draw a quadratic bezier curve between two points

```
void swf_shapecurveto (float x1, float y1, float x2, float y2)
```

The **swf_shapecurveto()** function draws a quadratic bezier curve from the x coordinate given by *x1* and the y coordinate given by *y1* to the x coordinate given by *x2* and the y coordinate given by *y2*. The current position is then set to the x,y coordinates given by the *x2* and *y2* parameters

swf_shapecurveto3 (PHP 4 >= 4.0RC2)

Draw a cubic bezier curve

```
void swf_shapecurveto3 (float x1, float y1, float x2, float y2, float x3, float y3)
```

Draw a cubic bezier curve using the x,y coordinate pairs *x1*,*y1* and *x2*,*y2* as off curve control points and the x,y coordinate *x3*, *y3* as an endpoint. The current position is then set to the x,y coordinate pair given by *x3*,*y3*.

swf_shapearc (PHP 4 >= 4.0RC2)

Draw a circular arc

```
void swf_shapearc (float x, float y, float r, float ang1, float ang2)
```

The **swf_shapeArc()** function draws a circular arc from angle A given by the *ang1* parameter to angle B given by the *ang2* parameter. The center of the circle has an x coordinate given by the *x* parameter and a y coordinate given by the *y*, the radius of the circle is given by the *r* parameter.

swf_endshape (PHP 4 >= 4.0RC2)

Completes the definition of the current shape

```
void swf_endshape (void);
```

The **swf_endshape()** completes the definition of the current shape.

swf_definefont (PHP 4 >= 4.0RC2)

Defines a font

```
void swf_definefont (int fontid, string fontname)
```

The **swf_definefont()** function defines a font given by the *fontname* parameter and gives it the id specified by the *fontid* parameter. It then sets the font given by *fontname* to the current font.

swf_setfont (PHP 4 >= 4.0RC2)

Change the current font

```
void swf_setfont (int fontid)
```

The **swf_setfont()** sets the current font to the value given by the *fontid* parameter.

swf_fontsize (PHP 4 >= 4.0RC2)

Change the font size

```
void swf_fontsize (float size)
```

The **swf_fontsize()** function changes the font size to the value given by the *size* parameter.

swf_fontslant (PHP 4 >= 4.0RC2)

Set the font slant

```
void swf_fontslant (float slant)
```

Set the current font slant to the angle indicated by the *slant* parameter. Positive values create a foward slant, negative values create a negative slant.

swf_fonttracking (PHP 4 >= 4.0RC2)

Set the current font tracking

```
void swf_fonttracking (float tracking)
```

Set the font tracking to the value specified by the *tracking* parameter. This function is used to increase the spacing between letters and text, positive values increase the space and negative values decrease the space between letters.

swf_getfontinfo (PHP 4 >= 4.0RC2)

The height in pixels of a capital A and a lowercase x

```
array swf_getfontinfo (void);
```

The **swf_getfontinfo()** function returns an associative array with the following parameters:

- *Aheight* - The height in pixels of a capital A.
- *xheight* - The height in pixels of a lowercase x.

swf_definetext (PHP 4 >= 4.0RC2)

Define a text string

```
void swf_definetext (int objid, string str, int docenter)
```

Define a text string (the *str* parameter) using the current font and font size. The *docenter* is where the word is centered, if *docenter* is 1, then the word is centered in x.

swf_textwidth (PHP 4 >= 4.0RC2)

Get the width of a string

```
float swf_textwidth (string str)
```

The **swf_textwidth()** function gives the width of the string, *str*, in pixels, using the current font and font size.

swf_definebitmap (PHP 4 >= 4.0RC2)

Define a bitmap

```
void swf_definebitmap (int objid, string image_name)
```

The **swf_definebitmap()** function defines a bitmap given a GIF, JPEG, RGB or FI image. The image will be converted into a Flash JPEG or Flash color map format.

swf_getbitmapinfo (PHP 4 >= 4.0RC2)

Get information about a bitmap

```
array swf_getbitmapinfo (int bitmapid)
```

The **swf_getbitmapinfo()** function returns an array of information about a bitmap given by the *bitmapid* parameter. The returned array has the following elements:

- "size" - The size in bytes of the bitmap.
- "width" - The width in pixels of the bitmap.
- "height" - The height in pixels of the bitmap.

swf_startsymbol (PHP 4 >= 4.0RC2)

Define a symbol

```
void swf_startsymbol (int objid)
```

Define an object id as a symbol. Symbols are tiny flash movies that can be played simultaneously. The *objid* parameter is the object id you want to define as a symbol.

swf_endsymbol (PHP 4 >= 4.0RC2)

End the definition of a symbol

```
void swf_endsymbol (void);
```

The **swf_endsymbol()** function ends the definition of a symbol that was started by the **swf_startsymbol()** function.

swf_startbutton (PHP 4 >= 4.0RC2)

Start the definition of a button

```
void swf_startbutton (int objid, int type)
```

The **swf_startbutton()** function starts off the definition of a button. The *type* parameter can either be TYPE_MENUBUTTON or TYPE_PUSHBUTTON. The TYPE_MENUBUTTON constant allows the focus to travel from the button when the mouse is down, TYPE_PUSHBUTTON does not allow the focus to travel when the mouse is down.

swf_addbuttonrecord (PHP 4 >= 4.0RC2)

Controls location, appearance and active area of the current button

```
void swf_addbuttonrecord (int states, int shapeid, int depth)
```

The **swf_addbuttonrecord()** function allows you to define the specifics of using a button. The first parameter, *states*, defines what states the button can have, these can be any or all of the following constants: BSHitTest, BSDown, BSOver or BSUp. The second parameter, the *shapeid* is the look of the button, this is usually the object id of the shape of the button. The *depth* parameter is the placement of the button in the current frame.

Example 1. Swf_addbuttonrecord() function example

```
swf_startButton ($objid, TYPE_MENUBUTTON);
    swf_addButtonRecord (BSDown|BSOver, $buttonImageId, 340);
    swf_onCondition (MenuEnter);
        swf_actionGetUrl ("http://www.designmultimedia.com", "_level1");
    swf_onCondition (MenuExit);
        swf_actionGetUrl ("", "_level1");
swf_endButton ();
```

swf_oncondition (PHP 4 >= 4.0RC2)

Describe a transition used to trigger an action list

```
void swf_oncondition (int transition)
```

The **swf_onCondition()** function describes a transition that will trigger an action list. There are several types of possible transitions, the following are for buttons defined as TYPE_MENUBUTTON:

- IdletoOverUp
- OverUptoIdle
- OverUptoOverDown
- OverDowntoOverUp
- IdletoOverDown
- OutDowntoIdle
- MenuEnter (IdletoOverUp|IdletoOverDown)
- MenuExit (OverUptoIdle|OverDowntoIdle)

For TYPE_PUSHBUTTON there are the following options:

- IdletoOverUp
- OverUptoIdle
- OverUptoOverDown
- OverDowntoOverUp
- OverDowntoOutDown
- OutDowntoOverDown

- OutDowntoIdle
- ButtonEnter (IdletoOverUp|OutDowntoOverDown)
- ButtonExit (OverUptoIdle|OverDowntoOutDown)

swf_endbutton (PHP 4 >= 4.0RC2)

End the definition of the current button

```
void swf_endbutton (void);
```

The **swf_endButton()** function ends the definition of the current button.

swf_viewport (PHP 4 >= 4.0RC2)

Select an area for future drawing

```
void swf_viewport (double xmin, double xmax, double ymin, double ymax)
```

The **swf_viewport()** function selects an area for future drawing for *xmin* to *xmax* and *ymin* to *ymax*, if this function is not called the area defaults to the size of the screen.

swf_ortho (PHP 4 >= 4.0.1)

Defines an orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)
```

The **swf_ortho()** function defines a orthographic mapping of user coordinates onto the current viewport.

swf_ortho2 (PHP 4 >= 4.0RC2)

Defines 2D orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho2 (double xmin, double xmax, double ymin, double ymax)
```

The **swf_ortho2()** function defines a two dimensional orthographic mapping of user coordinates onto the current viewport, this defaults to one to one mapping of the area of the Flash movie. If a perspective transformation is desired, the **swf_perspective ()** function can be used.

swf_perspective (PHP 4 >= 4.0RC2)

Define a perspective projection transformation

```
void swf_perspective (double fovy, double aspect, double near, double far)
```

The **swf_perspective()** function defines a perspective projection transformation. The *fovy* parameter is field-of-view angle in the y direction. The *aspect* parameter should be set to the aspect ratio of the viewport that is being drawn onto. The *near* parameter is the near clipping plane and the *far* parameter is the far clipping plane.

Note: Various distortion artifacts may appear when performing a perspective projection, this is because Flash players only have a two dimensional matrix. Some are not too pretty.

swf_polarview (PHP 4 >= 4.0RC2)

Define the viewer's position with polar coordinates

```
void swf_polarview (double dist, double azimuth, double incidence, double twist)
```

The **swf_polarview()** function defines the viewer's position in polar coordinates. The *dist* parameter gives the distance between the viewpoint to the world space origin. The *azimuth* parameter defines the azimuthal angle in the x,y coordinate plane, measured in distance from the y axis. The *incidence* parameter defines the angle of incidence in the y,z plane, measured in distance from the z axis. The incidence angle is defined as the angle of the viewport relative to the z axis. Finally the *twist* specifies the amount that the viewpoint is to be rotated about the line of sight using the right hand rule.

swf_lookat (PHP 4 >= 4.0RC2)

Define a viewing transformation

```
void swf_lookat (double view_x, double view_y, double view_z, double reference_x, double reference_y, double reference_z, double twist)
```

The **swf_lookat()** function defines a viewing transformation by giving the viewing position (the parameters *view_x*, *view_y*, and *view_z*) and the coordinates of a reference point in the scene, the reference point is defined by the *reference_x*, *reference_y* , and *reference_z* parameters. The *twist* controls the rotation along with viewer's z axis.

swf_pushmatrix (PHP 4 >= 4.0RC2)

Push the current transformation matrix back unto the stack

```
void swf_pushmatrix (void);
```

The **swf_pushmatrix()** function pushes the current transformation matrix back onto the stack.

swf_popmatrix (PHP 4 >= 4.0RC2)

Restore a previous transformation matrix

```
void swf_popmatrix (void);
```

The **swf_popmatrix()** function pushes the current transformation matrix back onto the stack.

swf_scale (PHP 4 >= 4.0RC2)

Scale the current transformation

```
void swf_scale (double x, double y, double z)
```

The **swf_scale()** scales the x coordinate of the curve by the value of the *x* parameter, the y coordinate of the curve by the value of the *y* parameter, and the z coordinate of the curve by the value of the *z* parameter.

swf_translate (PHP 4 >= 4.0RC2)

Translate the current transformations

```
void swf_translate (double x, double y, double z)
```

The **swf_translate()** function translates the current transformation by the *x*, *y*, and *z* values given.

swf_rotate (PHP 4 >= 4.0RC2)

Rotate the current transformation

```
void swf_rotate (double angle, string axis)
```

The **swf_rotate()** rotates the current transformation by the angle given by the *angle* parameter around the axis given by the *axis* parameter. Valid values for the axis are 'x' (the x axis), 'y' (the y axis) or 'z' (the z axis).

swf_posround (PHP 4 >= 4.0RC2)

Enables or Disables the rounding of the translation when objects are placed or moved

```
void swf_posround (int round)
```

The **swf_posround()** function enables or disables the rounding of the translation when objects are placed or moved, there are times when text becomes more readable because rounding has been enabled. The *round* is whether to enable rounding or not, if set to the value of 1, then rounding is enabled, if set to 0 then rounding is disabled.

LXVIII. SNMP functions

In order to use the SNMP functions on Unix you need to install the UCD SNMP (<http://ucd-snmp.ucdavis.edu/>) package.
On Windows these functions are only available on NT and not on Win95/98.

Important: In order to use the UCD SNMP package, you need to define NO_ZEROLENGTH_COMMUNITY to 1 before compiling it. After configuring UCD SNMP, edit config.h and search for NO_ZEROLENGTH_COMMUNITY.
Uncomment the #define line. It should look like this afterwards:

```
#define NO_ZEROLENGTH_COMMUNITY 1
```

If you see strange segmentation faults in combination with SNMP commands, you did not follow the above instructions. If you do not want to recompile UCD SNMP, you can compile PHP with the –enable-ucd-snmp-hack switch which will work around the misfeature.

snmpget (PHP 3, PHP 4)

Fetch an SNMP object

```
string snmpget (string hostname, string community, string object_id [, int timeout [, int retries]])
```

Returns SNMP object value on success and false on error.

The **snmpget()** function is used to read the value of an SNMP object specified by the *object_id*. SNMP agent is specified by the *hostname* and the read community is specified by the *community* parameter.

```
$syscontact = snmpget("127.0.0.1", "public", "system.SysContact.0");
```

snmpset (PHP 3>= 3.0.12, PHP 4 >= 4.0b2)

Set an SNMP object

```
bool snmpset (string hostname, string community, string object_id, string type, mixed value [, int timeout [, int retries]])
```

Sets the specified SNMP object value, returning true on success and false on error.

The **snmpset()** function is used to set the value of an SNMP object specified by the *object_id*. SNMP agent is specified by the *hostname* and the read community is specified by the *community* parameter.

snmpwalk (PHP 3, PHP 4)

Fetch all the SNMP objects from an agent

```
array snmpwalk (string hostname, string community, string object_id [, int timeout [, int retries]])
```

Returns an array of SNMP object values starting from the **object_id()** as root and false on error.

snmpwalk() function is used to read all the values from an SNMP agent specified by the *hostname*. *Community* specifies the read community for that agent. A null *object_id* is taken as the root of the SNMP objects tree and all objects under that tree are returned as an array. If *object_id* is specified, all the SNMP objects below that *object_id* are returned.

```
$a = snmpwalk("127.0.0.1", "public", "");
```

Above function call would return all the SNMP objects from the SNMP agent running on localhost. One can step through the values with a loop

```
for ($i=0; $i<count($a); $i++) {  
    echo $a[$i];  
}
```

snmpwalkoid (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Query for a tree of information about a network entity

```
array snmpwalkoid (string hostname, string community, string object_id [, int timeout [, int retries]])
```

Returns an associative array with object ids and their respective object value starting from the *object_id* as root and false on error.

snmpwalkoid() function is used to read all object ids and their respective values from an SNMP agent specified by the hostname. Community specifies the read *community* for that agent. A null *object_id* is taken as the root of the SNMP objects tree and all objects under that tree are returned as an array. If *object_id* is specified, all the SNMP objects below that *object_id* are returned.

The existence of **snmpwalkoid()** and **snmpwalk()** has historical reasons. Both functions are provided for backward compatibility.

```
$a = snmpwalkoid("127.0.0.1", "public", "");
```

Above function call would return all the SNMP objects from the SNMP agent running on localhost. One can step through the values with a loop

```
for (reset($a); $i = key($a); next($a)) {
    echo "$i: $a[$i]<br>\n";
}
```

snmp_get_quick_print (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Fetch the current value of the UCD library's quick_print setting

```
boolean snmp_get_quick_print (void )
```

Returns the current value stored in the UCD Library for quick_print. quick_print is off by default.

```
$quickprint = snmp_get_quick_print();
```

Above function call would return false if quick_print is on, and true if quick_print is on.

snmp_get_quick_print() is only available when using the UCD SNMP library. This function is not available when using the Windows SNMP library.

See: **snmp_set_quick_print()** for a full description of what quick_print does.

snmp_set_quick_print (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Set the value of quick_print within the UCD SNMP library.

```
void snmp_set_quick_print (boolean quick_print)
```

Sets the value of quick_print within the UCD SNMP library. When this is set (1), the SNMP library will return 'quick printed' values. This means that just the value will be printed. When quick_print is not enabled (default) the UCD SNMP library prints extra information including the type of the value (i.e. IpAddress or OID). Additionally, if quick_print is not enabled, the library prints additional hex values for all strings of three characters or less.

Setting quick_print is often used when using the information returned rather than displaying it.

```
snmp_set_quick_print(0);
$a = snmpget("127.0.0.1", "public", ".1.3.6.1.2.1.2.2.1.9.1");
echo "$a<BR>\n";
snmp_set_quick_print(1);
$a = snmpget("127.0.0.1", "public", ".1.3.6.1.2.1.2.2.1.9.1");
echo "$a<BR>\n";
```

The first value printed might be: 'Timeticks: (0) 0:00:00.00', whereas with quick_print enabled, just '0:00:00.00' would be printed.

By default the UCD SNMP library returns verbose values, quick_print is used to return only the value.

Currently strings are still returned with extra quotes, this will be corrected in a later release.

snmp_set_quick_print() is only available when using the UCD SNMP library. This function is not available when using the Windows SNMP library.

LXIX. Socket functions

The socket extension implements a low-level interface to the socket communication functions, providing the possibility to act as a socket server as well as a client.

The socket functions described here are part of an extension to PHP which must be enabled at compile time by giving the `-enable-sockets` option to **configure**.

For a more generic client-side socket interface, see **fsockopen()** and **pfsockopen()**.

When using these functions, it is important to remember that while many of them have identical names to their C counterparts, they often have different declarations. Please be sure to read the descriptions to avoid confusion.

That said, those unfamiliar with socket programming can still find a lot of useful material in the appropriate Unix man pages, and there is a great deal of tutorial information on socket programming in C on the web, much of which can be applied, with slight modifications, to socket programming in PHP.

Example 1. Socket example: Simple TCP/IP server

This example shows a simple talkback server. Change the `$address` and `$port` variables to suit your setup and execute. You may then connect to the server with a command similar to: **telnet 192.168.1.53 10000** (where the address and port match your setup). Anything you type will then be output on the server side, and echoed back to you. To disconnect, enter 'quit'.

```
<?php
error_reporting (E_ALL);

/* Allow the script to hang around waiting for connections. */
set_time_limit (0);

$address = '192.168.1.53';
$port = 10000;

if (($sock = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
    echo "socket() failed: reason: " . strerror ($sock) . "\n";
}

if (($ret = bind ($sock, $address, $port)) < 0) {
    echo "bind() failed: reason: " . strerror ($ret) . "\n";
}

if (($ret = listen ($sock, 5)) < 0) {
    echo "listen() failed: reason: " . strerror ($ret) . "\n";
}

do {
    if (($msgsock = accept_connect($sock)) < 0) {
        echo "accept_connect() failed: reason: " . strerror ($msgsock) . "\n";
        break;
    }
    do {
        $buf = "";
        $ret = read ($msgsock, $buf, 2048);
        if ($ret < 0) {
            echo "read() failed: reason: " . strerror ($ret) . "\n";
            break 2;
        }
        if ($ret == 0) {
            break 2;
        }
        $buf = trim ($buf);
        if ($buf == 'quit') {
            close ($msgsock);
        }
    }
}
```

```

        break 2;
    }
    $talkback = "PHP: You said '$buf'.\n";
    write ($msgsock, $talkback, strlen ($talkback));
    echo "$buf\n";
} while (true);
close ($msgsock);
} while (true);

close ($sock);
?>

```

Example 2. Socket example: Simple TCP/IP client

This example shows a simple, one-shot HTTP client. It simply connects to a page, submits a HEAD request, echoes the reply, and exits.

```

<?php
error_reporting (E_ALL);

echo "<h2>TCP/IP Connection</h2>\n";

/* Get the port for the WWW service. */
$service_port = getservbyname ('www', 'tcp');

/* Get the IP address for the target host. */
$address = gethostbyname ('www.php.net');

/* Create a TCP/IP socket. */
$socket = socket (AF_INET, SOCK_STREAM, 0);
if ($socket < 0) {
    echo "socket() failed: reason: " . strerror ($socket) . "\n";
} else {
    "socket() successful: " . strerror ($socket) . "\n";
}

echo "Attempting to connect to '$address' on port '$service_port'...";
$result = connect ($socket, $address, $service_port);
if ($result < 0) {
    echo "connect() failed.\nReason: ($result) " . strerror($result) . "\n";
} else {
    echo "OK.\n";
}

$in = "HEAD / HTTP/1.0\r\n\r\n";
$out = "";

echo "Sending HTTP HEAD request...";
write ($socket, $in, strlen ($in));
echo "OK.\n";

echo "Reading response:\n\n";
while (read ($socket, $out, 2048)) {
    echo $out;
}

echo "Closing socket...";
close ($socket);
echo "OK.\n\n";

```

?>

accept_connect (PHP 4 >= 4.0.2)

Accepts a connection on a socket

```
int accept_connect (int socket)
```

After the socket *socket* has been created using **socket()**, bound to a name with **bind()**, and told to listen for connections with **listen()**, this function will accept incoming connections on that socket. Once a successful connection is made, a new socket descriptor is returned, which may be used for communication. If there are multiple connections queued on the socket, the first will be used. If there are no pending connections, **accept_connect()** will block until a connection becomes present. If *socket* has been made non-blocking using **socket_set_blocking()** or **set_nonblock()**, an error code will be returned.

The socket descriptor returned by **accept_connect()** may not be used to accept new connections. The original listening socket *socket*, however, remains open and may be reused.

Returns a new socket descriptor on success, or a negative error code on failure. This code may be passed to **strerror()** to get a textual explanation of the error.

See also **bind()**, **connect()**, **listen()**, **socket()**, **socket_get_status()**, and **strerror()**.

bind (PHP 4 >= 4.0.2)

Binds a name to a socket

```
int bind (int socket, string address [, int port])
```

bind() binds the name given in *address* to the socket described by *socket*, which must be a valid socket descriptor created with **socket()**.

The *address* parameter is either an IP address in dotted-quad notation (e.g. 127.0.0.1), if the socket is of the AF_INET family; or the pathname of a Unix-domain socket, if the socket family is AF_UNIX.

The *port* parameter is only used when connecting to an AF_INET socket, and designates the port on the remote host to which a connection should be made.

Returns zero on success, or a negative error code on failure. This code may be passed to **strerror()** to get a textual explanation of the error.

See also **accept_connect()**, **connect()**, **listen()**, **socket()**, **socket_get_status()**, and **strerror()**.

close (PHP 4)

Closes a file descriptor

```
bool close (int socket)
```

close() closes the file (or socket) descriptor given by *socket*.

Note that **close()** should not be used on PHP file descriptors created with **fopen()**, **popen()**, **fsckopen()**, or **psockopen()**; it is meant for sockets created with **socket()** or **accept_connect()**.

Returns true on success, or false if an error occurs (i.e., *socket* is invalid).

See also **bind()**, **listen()**, **socket()**, **socket_get_status()**, and **strerror()**.

connect (PHP 4 >= 4.0.2)

Initiates a connection on a socket

```
int connect (int socket, string address [, int port])
```

Initiates a connection using the socket descriptor *socket*, which must be a valid socket descriptor created with **socket()**.

The *address* parameter is either an IP address in dotted-quad notation (e.g. 127.0.0.1), if the socket is of the AF_INET family; or the pathname of a Unix-domain socket, if the socket family is AF_UNIX.

The *port* parameter is only used when connecting to an AF_INET socket, and designates the port on the remote host to which a connection should be made.

Returns zero on success, or a negative error code on failure. This code may be passed to **strerror()** to get a textual explanation of the error.

See also **bind()**, **listen()**, **socket()**, **socket_get_status()**, and **strerror()**.

listen (PHP 4 >= 4.0.2)

Listens for a connection on a socket

```
int listen (int socket, int backlog)
```

After the socket *socket* has been created using **socket()** and bound to a name with **bind()**, it may be told to listen for incoming connections on *socket*. A maximum of *backlog* incoming connections will be queued for processing.

listen() is applicable only to sockets with type SOCK_STREAM or SOCK_SEQPACKET.

Returns zero on success, or a negative error code on failure. This code may be passed to **strerror()** to get a textual explanation of the error.

See also **accept_connect()**, **bind()**, **connect()**, **socket()**, **socket_get_status()**, and **strerror()**.

read (PHP 4)

Read from a socket

```
int read (int socket_des, string &buffer, int length)
```

The function **read()** reads from socket *socket_des* created by the **accept_connect()** function into *&buffer* the number of bytes set by *length*. Otherwise you can use \n, \t or \0 to end reading. Returns number of bytes that have been read.

See also **accept_connect()**, **bind()**, **connect()**, **listen()**, **strerror()**, **socket_get_status()**, and **write()**.

socket (PHP 4 >= 4.0.2)

Create a socket (endpoint for communication)

```
int socket (int domain, int type, int protocol)
```

Creates a communication endpoint (a socket), and returns a descriptor to the socket.

The *domain* parameter sets the domain. Currently, `AF_INET` and `AF_UNIX` are understood.

The *type* parameter selects the socket type. This is one of `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_SEQPACKET`, `SOCK_RAW`, `SOCK_RDM`, or `SOCK_PACKET`.

protocol sets the protocol.

Returns a valid socket descriptor on success, or a negative error code on failure. This code may be passed to `strerror()` to get a textual explanation of the error.

For more information on the usage of `socket()`, as well as on the meanings of the various parameters, see the Unix man page `socket` (2).

See also `accept_connect()`, `bind()`, `connect()`, `listen()`, `strerror()`, and `socket_get_status()`.

strerror (PHP 4 >= 4.0.2)

Return a string describing a socket error

```
string strerror (int errno)
```

`strerror()` takes as its *errno* parameter the return value of one of the socket functions, and returns the corresponding explanatory text. This makes it a bit more pleasant to figure out why something didn't work; for instance, instead of having to track down a system include file to find out what '-111' means, you just pass it to `strerror()`, and it tells you what happened.

Example 1. `strerror()` example

```
<?php
if (($socket = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
    echo "socket() failed: reason: " . strerror ($socket) . "\n";
}

if (($ret = bind ($socket, '127.0.0.1', 80)) < 0) {
    echo "bind() failed: reason: " . strerror ($ret) . "\n";
}
?>
```

The expected output from the above example (assuming the script is not run with root privileges):

```
bind() failed: reason: Permission denied
```

See also `accept_connect()`, `bind()`, `connect()`, `listen()`, `socket()`, and `socket_get_status()`.

write (PHP 4 >= 4.0.2)

Write to a socket

```
int write (int socket_des, string &buffer, int length)
```

The function `write()` writes to the socket *socket_des* from *buffer* the number of bytes set by *length*.

See also `accept_connect()`, `bind()`, `connect()`, `listen()`, `read()`, `strerror()`, and `socket_get_status()`.

LXX. String functions

These functions all manipulate strings in various ways. Some more specialized sections can be found in the regular expression and URL handling sections.

For information on how strings behave, especially with regard to usage of single quotes, double quotes, and escape sequences, see the [Strings](#) entry in the [Types](#) section of the manual.

AddCSlashes (PHP 4 >= 4.0b4)

Quote string with slashes in a C style

```
string addcslashes (string str, string charlist)
```

Returns a string with backslashes before characters that are listed in *charlist* parameter. It escapes \n, \r etc. in C-like style, characters with ASCII code lower than 32 and higher than 126 are converted to octal representation. Be carefull when escaping alphanumeric characters. You can specify a range in *charlist* like "\0..\37", which would escape all characters with ASCII code between 0 and 31.

Example 1. Addcslashes() example

```
$escaped = addcslashes ($not_escaped, "\0..\37!@\177..\377");
```

Note: Added in PHP4b3-dev.

See also **stripcslashes()**, **stripslashes()**, **htmlspecialchars()**, and **quotemeta()**.

AddSlashes (PHP 3, PHP 4)

Quote string with slashes

```
string addslashes (string str)
```

Returns a string with backslashes before characters that need to be quoted in database queries etc. These characters are single quote ('), double quote ("), backslash (\) and NUL (the null byte).

See also **stripslashes()**, **htmlspecialchars()**, and **quotemeta()**.

bin2hex (PHP 3>= 3.0.9, PHP 4)

Convert binary data into hexadecimal representation

```
string bin2hex (string str)
```

Returns an ASCII string containing the hexadecimal representation of *str*. The conversion is done byte-wise with the high-nibble first.

Chop (PHP 3, PHP 4)

Remove trailing whitespace

```
string chop (string str)
```

Returns the argument string without trailing whitespace, including newlines.

Example 1. Chop() example

```
$trimmed = chop ($line);
```

Note: **chop()** is different than the Perl *chop()* function, which removes the last character in the string.

See also **trim()**, **ltrim()**, **rtrim()**, and **chop()**.

Chr (PHP 3, PHP 4)

Return a specific character

```
string chr (int ascii)
```

Returns a one-character string containing the character specified by *ascii*.

Example 1. Chr() example

```
$str .= chr (27); /* add an escape character at the end of $str */
/* Often this is more useful */

$str = sprintf ("The string ends in escape: %c", 27);
```

This function complements **ord()**. See also **sprintf()** with a format string of %c.

chunk_split (PHP 3>= 3.0.6, PHP 4)

Split a string into smaller chunks

```
string chunk_split (string string [, int chunklen [, string end]])
```

Can be used to split a string into smaller chunks which is useful for e.g. converting **base64_encode** output to match RFC 2045 semantics. It inserts every *chunklen* (defaults to 76) chars the string *end* (defaults to "\r\n"). It returns the new string leaving the original string untouched.

Example 1. Chunk_split() example

```
# format $data using RFC 2045 semantics
$new_string = chunk_split (base64_encode($data));
```

This function is significantly faster than **ereg_replace()**.

Note: This function was added in 3.0.6.

convert_cyr_string (PHP 3>= 3.0.6, PHP 4)

Convert from one Cyrillic character set to another

```
string convert_cyr_string (string str, string from, string to)
```

This function converts the given string from one Cyrillic character set to another. The *from* and *to* arguments are single characters that represent the source and target Cyrillic character sets. The supported types are:

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

count_chars (PHP 4 >= 4.0b4)

Return information abouts characters used in a string

```
mixed count_chars (string string [, mode])
```

Counts the number of occurrences of every byte-value (0..255) in *string* and returns it in various ways. The optional parameter *Mode* default to 0. Depending on *mode* **count_chars()** returns one of the following:

- 0 - an array with the byte-value as key and the frequency of every byte as value.
- 1 - same as 0 but only byte-values with a frequency greater than zero are listed.
- 2 - same as 0 but only byte-values with a frequency equal to zero are listed.
- 3 - a string containing all used byte-values is returned.
- 4 - a string containing all not used byte-values is returned.

Note: This function was added in PHP 4.0.

crc32 (PHP 4 >= 4.0.1)

Calculates the crc32 polynomial of a string

```
int crc32 (string str)
```

Generates the cyclic redundancy checksum polynomial of 32-bit lengths of the *str*. This is usually used to validate the integrity of data being trasmited.

See also: **md5()**

crypt (PHP 3, PHP 4)

DES-encrypt a string

```
string crypt (string str [, string salt])
```

crypt() will encrypt a string using the standard Unix DES encryption method. Arguments are a string to be encrypted and an optional two-character salt string to base the encryption on. See the Unix man page for your crypt function for more information.

If the salt argument is not provided, one will be randomly generated by PHP.

Some operating systems support more than one type of encryption. In fact, sometimes the standard DES encryption is replaced by an MD5 based encryption algorithm. The encryption type is triggered by the salt argument. At install time, PHP determines the capabilities of the crypt function and will accept salts for other encryption types. If no salt is provided, PHP will auto-generate a standard 2-character DES salt by default, unless the default encryption type on the system is MD5, in which case a random MD5-compatible salt is generated. PHP sets a constant named CRYPT_SALT_LENGTH which tells you whether a regular 2-character salt applies to your system or the longer 12-char MD5 salt is applicable.

If you are using the supplied salt, you should be aware that the salt is generated once. If you are calling this function recursively, this may impact both appearance and, to a certain extent, security.

The standard DES encryption **crypt()** contains the salt as the first two characters of the output.

On systems where the **crypt()** function supports multiple encryption types, the following constants are set to 0 or 1 depending on whether the given type is available:

- CRYPT_STD_DES - Standard DES encryption with a 2-char SALT
- CRYPT_EXT_DES - Extended DES encryption with a 9-char SALT
- CRYPT_MD5 - MD5 encryption with a 12-char SALT starting with \$1\$
- CRYPT_BLOWFISH - Extended DES encryption with a 16-char SALT starting with \$2\$

There is no decrypt function, since **crypt()** uses a one-way algorithm.

See also: **md5()**.

echo (unknown)

Output one or more strings

```
echo (string arg1, string [argn]...)
```

Outputs all parameters.

Echo() is not actually a function (it is a language construct) so you are not required to use parentheses with it.

Example 1. Echo() example

```
echo "Hello World";  
  
echo "This spans  
multiple lines. The newlines will be  
output as well";  
  
echo "This spans\nmultiple lines. The newlines will be\noutput as well.";
```

Note: In fact, if you want to pass more than one parameter to echo, you must not enclose the parameters within parentheses.

See also: **print()**, **printf()**, and **flush()**.

explode (PHP 3, PHP 4)

Split a string by string

```
array explode (string separator, string string [, int limit])
```

Returns an array of strings, each of which is a substring of *string* formed by splitting it on boundaries formed by the string *delim*. If *limit* is set, the returned array will contain a maximum of *limit* elements with the last element containing the whole rest of *string*.

Note: The *limit* parameter was added in PHP 4.0.1

Example 1. Explode() example

```
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode (" ", $pizza);
```

Note: Although **implode()** can for historical reasons accept its parameters in either order, **explode()** cannot. You must ensure that the *separator* argument comes before the *string* argument.

See also **split()** and **implode()**.

get_html_translation_table (PHP 4 >= 4.0b4)

Returns the translation table used by **htmlspecialchars()** and **htmlentities()**

```
string get_html_translation_table (int table [, int quote_style])
```

get_html_translation_table() will return the translation table that is used internally for **htmlspecialchars()** and **htmlentities()**. There are two new defines (*HTML_ENTITIES*, *HTML_SPECIALCHARS*) that allow you to specify the table you want. And as in the **htmlspecialchars()** and **htmlentities()** functions you can optionally specify the *quote_style* you are working with. The default is ENT_COMPAT mode. See the description of these modes in **htmlspecialchars()**.

Example 1. Translation Table Example

```
$trans = get_html_translation_table (HTML_ENTITIES);
$str = "Hallo & <Frau> & Krämer";
$encoded = strtr ($str, $trans);
```

The *encoded* variable will now contain: "Hallo & <Frau> & Krämer".

The cool thing is using **array_flip()** to change the direction of the translation.

```
$trans = array_flip ($trans);
$original = strtr ($str, $trans);
```

The content of `$original` would be: "Hallo & <Frau> & Krämer".

Note: This function was added in PHP 4.0.

See also: **htmlspecialchars()**, **htmlentities()**, **strtr()**, and **array_flip()**.

get_meta_tags (PHP 3>= 3.0.4, PHP 4)

Extracts all meta tag content attributes from a file and returns an array

```
array get_meta_tags (string filename [, int use_include_path])
```

Opens `filename` and parses it line by line for <meta> tags of the form

Example 1. Meta Tags Example

```
<meta name="author" content="name">
<meta name="tags" content="php3 documentation">
</head> <!-- parsing stops here -->
```

(pay attention to line endings - PHP uses a native function to parse the input, so a Mac file won't work on Unix).

The value of the name property becomes the key, the value of the content property becomes the value of the returned array, so you can easily use standard array functions to traverse it or access single values. Special characters in the value of the name property are substituted with '_', the rest is converted to lower case.

Setting `use_include_path` to 1 will result in PHP trying to open the file along the standard include path.

hebrev (PHP 3, PHP 4)

Convert logical Hebrew text to visual text

```
string hebrev (string hebrew_text [, int max_chars_per_line])
```

The optional parameter `max_chars_per_line` indicates maximum number of characters per line will be output. The function tries to avoid breaking words.

See also **hebrevc()**

hebrevc (PHP 3, PHP 4)

Convert logical Hebrew text to visual text with newline conversion

```
string hebrevc (string hebrew_text [, int max_chars_per_line])
```

This function is similar to **hebrev()** with the difference that it converts newlines (\n) to "
\n". The optional parameter *max_chars_per_line* indicates maximum number of characters per line will be output. The function tries to avoid breaking words.

See also **hebrev()**

htmlentities (PHP 3, PHP 4)

Convert all applicable characters to HTML entities

```
string htmlentities (string string [, int quote_style])
```

This function is identical to **htmlspecialchars()** in all ways, except that all characters which have HTML character entity equivalents are translated into these entities. Like **htmlspecialchars()**, it takes an optional second argument which indicates what should be done with single and double quotes. ENT_COMPAT (the default) will only convert double-quotes and leave single-quotes alone. ENT_QUOTES will convert both double and single quotes, and ENT_NOQUOTES will leave both double and single quotes unconverted.

At present, the ISO-8859-1 character set is used. Note that the optional second argument was added in PHP 3.0.17 and PHP 4.0.3.

See also **htmlspecialchars()** and **nl2br()**.

htmlspecialchars (PHP 3, PHP 4)

Convert special characters to HTML entities

```
string htmlspecialchars (string string [, int quote_style])
```

Certain characters have special significance in HTML, and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with some of these conversions made; the translations made are those most useful for everyday web programming. If you require all HTML character entities to be translated, use **htmlentities()** instead.

This function is useful in preventing user-supplied text from containing HTML markup, such as in a message board or guest book application. The optional second argument, *quote_style*, tells the function what to do with single and double quote characters. The default mode, ENT_COMPAT, is the backwards compatible mode which only translates the double-quote character and leaves the single-quote untranslated. If ENT_QUOTES is set, both single and double quotes are translated and if ENT_NOQUOTES is set neither single nor double quotes are translated.

The translations performed are:

- '&' (ampersand) becomes '&'
- '"' (double quote) becomes '"' when ENT_NOQUOTES is not set.
- "'" (single quote) becomes ''' only when ENT_QUOTES is set.
- '<' (less than) becomes '<'
- '>' (greater than) becomes '>'

Example 1. htmlspecialchars() example

```
$new = htmlspecialchars("<a href='test'>Test</a>" , ENT_QUOTES);
```

Note that this function does not translate anything beyond what is listed above. For full entity translation, see **htmlentities()**. Also note that the optional second argument was added in PHP 3.0.17 and PHP 4.0.3.

See also **htmlentities()** and **nl2br()**.

implode (PHP 3, PHP 4)

Join array elements with a string

```
string implode (string glue, array pieces)
```

Returns a string containing a string representation of all the array elements in the same order, with the glue string between each element.

Example 1. `Implode()` example

```
$colon_separated = implode ( ":" , $array );
```

Note: **implode()** can, for historical reasons, accept its parameters in either order. For consistency with **explode()**, however, it may be less confusing to use the documented order of arguments.

See also **explode()**, **join()**, and **split()**.

join (PHP 3, PHP 4)

Join array elements with a string

```
string join (string glue, array pieces)
```

join() is an alias to **implode()**, and is identical in every way.

See also **explode()**, **implode()**, and **split()**.

levenshtein (PHP 3>= 3.0.17, PHP 4 >= 4.0.1)

Calculate Levenshtein distance between two strings

```
int levenshtein (string str1, string str2)
int levenshtein (string str1, string str2, int cost_ins, int cost_rep, int cost_del)
int levenshtein (string str1, string str2, function cost)
```

This function returns the Levenshtein-Distance between the two argument strings or -1, if one of the argument strings is longer than the limit of 255 characters (255 should be more than enough for name or dictionary comparison, and nobody serious would be doing genetic analysis with PHP).

The Levenshtein distance is defined as the minimal number of characters you have to replace, insert or delete to transform *str1* into *str2*. The complexity of the algorithm is $O(m*n)$, where *n* and *m* are the length of *str1* and *str2* (rather good when compared to **similar_text()**, which is $O(\max(n,m)^{**}3)$, but still expensive).

In its simplest form the function will take only the two strings as parameter and will calculate just the number of insert, replace and delete operations needed to transform *str1* into *str2*.

A second variant will take three additional parameters that define the cost of insert, replace and delete operations. This is more general and adaptive than variant one, but not as efficient.

The third variant (which is not implemented yet) will be the most general and adaptive, but also the slowest alternative. It will call a user-supplied function that will determine the cost for every possible operation.

The user-supplied function will be called with the following arguments:

- operation to apply: 'I', 'R' or 'D'
- actual character in string 1
- actual character in string 2
- position in string 1
- position in string 2
- remaining characters in string 1
- remaining characters in string 2

The user-supplied function has to return a positive integer describing the cost for this particular operation, but it may decide to use only some of the supplied arguments.

The user-supplied function approach offers the possibility to take into account the relevance of and/or difference between certain symbols (characters) or even the context those symbols appear in to determine the cost of insert, replace and delete operations, but at the cost of losing all optimizations done regarding cpu register utilization and cache misses that have been worked into the other two variants.

See also **soundex()**, **similar_text()** and **metaphone()**.

localeconv (unknown)

Get numeric formatting information

```
array localeconv(void);
```

Returns an associative array containing localized numeric and monetary formatting information.

localeconv() returns data based upon the current locale as set by **setlocale()**. The associative array that is returned contains the following fields:

Array element	Description
decimal_point	Decimal point character
thousands_sep	Thousands separator
grouping	Array containing numeric groupings
int_curr_symbol	International currency symbol (i.e. USD)
currency_symbol	Local currency symbol (i.e. \$)
mon_decimal_point	Monetary decimal point character
mon_thousands_sep	Monetary thousands separator
mon_grouping	Array containing monetary groupings

Array element	Description
positive_sign	Sign for positive values
negative_sign	Sign for negative values
int_frac_digits	International fractional digits
frac_digits	Local fractional digits
p_cs_precedes	True if currency_symbol precedes a positive value, false if it succeeds one
p_sep_by_space	True if a space separates currency_symbol from a positive value, false otherwise
n_cs_precedes	True if currency_symbol precedes a negative value, false if it succeeds one
n_sep_by_space	True if a space separates currency_symbol from a negative value, false otherwise
p_sign_posn	0 Parentheses surround the quantity and currency_symbol 1 The sign string precedes the quantity and currency_symbol 2 The sign string succeeds the quantity and currency_symbol 3 The sign string immediately precedes the currency_symbol 4 The sign string immediately succeeds the currency_symbol
n_sign_posn	0 Parentheses surround the quantity and currency_symbol 1 The sign string precedes the quantity and currency_symbol 2 The sign string succeeds the quantity and currency_symbol 3 The sign string immediately precedes the currency_symbol 4 The sign string immediately succeeds the currency_symbol

The grouping fields contain arrays that define the way numbers should be grouped. For example, the grouping field for the en_US locale, would contain a 2 item array with the values 3 and 3. The higher the index in the array, the farther left the grouping is. If an array element is equal to CHAR_MAX, no further grouping is done. If an array element is equal to 0, the previous element should be used.

Example 1. `localeconv()` example

```
setlocale(LC_ALL, "en_US");

$locale_info = localeconv();

echo "<PRE>\n";
echo "-----\n";
echo " Monetary information for current locale: \n";
echo "-----\n\n";

echo "int_curr_symbol: {$locale_info["int_curr_symbol"]}\n";
```

```

echo "currency_symbol:      {$locale_info["currency_symbol"]}\n";
echo "mon_decimal_point:   {$locale_info["mon_decimal_point"]}\n";
echo "mon_thousands_sep:    {$locale_info["mon_thousands_sep"]}\n";
echo "positive_sign:        {$locale_info["positive_sign"]}\n";
echo "negative_sign:        {$locale_info["negative_sign"]}\n";
echo "int_frac_digits:      {$locale_info["int_frac_digits"]}\n";
echo "frac_digits:          {$locale_info["frac_digits"]}\n";
echo "p_cs_precedes:        {$locale_info["p_cs_precedes"]}\n";
echo "p_sep_by_space:       {$locale_info["p_sep_by_space"]}\n";
echo "n_cs_precedes:        {$locale_info["n_cs_precedes"]}\n";
echo "n_sep_by_space:       {$locale_info["n_sep_by_space"]}\n";
echo "p_sign_posn:          {$locale_info["p_sign_posn"]}\n";
echo "n_sign_posn:          {$locale_info["n_sign_posn"]}\n";
echo "</PRE>\n";

```

The constant CHAR_MAX is also defined for the use mentioned above.

Note: Added in PHP 4.0.5

See also: **setlocale()**.

Itrim

(PHP 3, PHP 4)

Strip whitespace from the beginning of a string

```
string ltrim (string str)
```

This function strips whitespace from the start of a string and returns the stripped string. The whitespace characters it currently strips are: "\n", "\r", "\t", "\v", "\0", and a plain space.

See also **chop()**, **rtrim()**, and **trim()**.

md5

(PHP 3, PHP 4)

Calculate the md5 hash of a string

```
string md5 (string str)
```

Calculates the MD5 hash of *str* using the RSA Data Security, Inc. MD5 Message-Digest Algorithm (<http://www.faqs.org/rfcs/rfc1321.html>).

See also: **crc32()**

Metaphone

(PHP 4 >= 4.0b4)

Calculate the metaphone key of a string

```
string metaphone (string str)
```

Calculates the metaphone key of *str*.

Similar to **soundex()** metaphone creates the same key for similar sounding words. It's more accurate than **soundex()** as it knows the basic rules of English pronunciation. The metaphone generated keys are of variable length.

Metaphone was developed by Lawrence Philips <lphilips@verity.com>. It is described in ["Practical Algorithms for Programmers", Binstock & Rex, Addison Wesley, 1995].

Note: This function was added in PHP 4.0.

nl2br (PHP 3, PHP 4)

Converts newlines to HTML line breaks

```
string nl2br (string string)
```

Returns *string* with '
' inserted before all newlines.

See also **htmlspecialchars()**, **htmlentities()** and **wordwrap()**.

Ord (PHP 3, PHP 4)

Return ASCII value of character

```
int ord (string string)
```

Returns the ASCII value of the first character of *string*. This function complements **chr()**.

Example 1. Ord() example

```
if (ord ($str) == 10) {
    echo "The first character of \$str is a line feed.\n";
}
```

See also **chr()**.

parse_str (PHP 3, PHP 4)

Parses the string into variables

```
void parse_str (string str [, array arr])
```

Parses *str* as if it were the query string passed via an URL and sets variables in the current scope. If the second parameter *arr* is present, variables are stored in this variable as an array elements instead.

Example 1. Using parse_str()

```
$str = "first=value&second[ ]=this+works&second[ ]=another";
parse_str($str);
```

```
echo $first;      /* prints "value" */
echo $second[0]; /* prints "this works" */
echo $second[1]; /* prints "another" */
```

print (unknown)

Output a string

```
print (string arg)
```

Outputs *arg*.

See also: **echo()**, **printf()**, and **flush()**.

printf (PHP 3, PHP 4)

Output a formatted string

```
int printf (string format [, mixed args...])
```

Produces output according to *format*, which is described in the documentation for **sprintf()**.

See also: **print()**, **sprintf()**, **sscanf()**, **fscanf()**, and **flush()**.

quoted_printable_decode (PHP 3>= 3.0.6, PHP 4)

Convert a quoted-printable string to an 8 bit string

```
string quoted_printable_decode (string str)
```

This function returns an 8-bit binary string corresponding to the decoded quoted printable string. This function is similar to **imap_qprint()**, except this one does not require the IMAP module to work.

quotemeta (PHP 3, PHP 4)

Quote meta characters

```
string quotemeta (string str)
```

Returns a version of str with a backslash character (\) before every character that is among these:

. \ \ + * ? [^] (\$)

See also **addslashes()**, **htmlentities()**, **htmlspecialchars()**, **nl2br()**, and **stripslashes()**.

rtrim (PHP 3, PHP 4)

Remove trailing whitespace.

```
string rtrim (string str)
```

Returns the argument string without trailing whitespace, including newlines. This is an alias for **chop()**.

Example 1. rtrim() example

```
$trimmed = rtrim ($line);
```

See also **trim()**, **ltrim()**, and **rtrim()**.

sscanf (PHP 4 >= 4.0.1)

Parses input from a string according to a format

```
mixed sscanf (string str, string format [, string var1...])
```

The function **sscanf()** is the input analog of **printf()**. **Sscanf()** reads from the string *str* and interprets it according to the specified *format*. If only two parameters were passed to this function, the values parsed will be returned as an array.

Example 1. Sscanf() Example

```
// getting the serial number
$serial = sscanf("SN/2350001","SN/%d");
// and the date of manufacturing
$mandate = "January 01 2000";
list($month, $day, $year) = sscanf($mandate,"%s %d %d");
echo "Item $serial was manufactured on: $year-".substr($month,0,3)."-".$day"\n";
```

If optional parameters are passed, the function will return the number of assigned values. The optional parameters must be passed by reference.

Example 2. Sscanf() - using optional parameters

```
// get author info and generate DocBook entry
$auth = "24\tLewis Carroll";
$n = sscanf($auth,"%d\t%s %s", &$id, &$first, &$last);
echo "<author id='\$id'>
<firstname>$first</firstname>
<surname>$last</surname>
</author>\n";
```

See also: **fscanf()**, **printf()**, and **sprintf()**.

setlocale (PHP 3, PHP 4)

Set locale information

```
string setlocale (string category, string locale)
```

Category is a string specifying the category of the functions affected by the locale setting:

- LC_ALL for all of the below
- LC_COLLATE for string comparison, see **strcoll()**
- LC_CTYPE for character classification and conversion, for example **strtoupper()**
- LC_MONETARY for **localeconv()**
- LC_NUMERIC for decimal separator (See also: **localeconv()**)
- LC_TIME for date and time formatting with **strftime()**

If *locale* is the empty string "", the locale names will be set from the values of environment variables with the same names as the above categories, or from "LANG".

If *locale* is zero or "0", the locale setting is not affected, only the current setting is returned.

Setlocale returns the new current locale, or false if the locale functionality is not implemented in the platform, the specified locale does not exist or the category name is invalid. An invalid category name also causes a warning message.

similar_text (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Calculate the similarity between two strings

```
int similar_text (string first, string second [, double percent])
```

This calculates the similarity between two strings as described in Oliver [1993]. Note that this implementation does not use a stack as in Oliver's pseudo code, but recursive calls which may or may not speed up the whole process. Note also that the complexity of this algorithm is O(N**3) where N is the length of the longest string.

By passing a reference as third argument, **similar_text()** will calculate the similarity in percent for you. It returns the number of matching chars in both strings.

soundex (PHP 3, PHP 4)

Calculate the soundex key of a string

```
string soundex (string str)
```

Calculates the soundex key of *str*.

Soundex keys have the property that words pronounced similarly produce the same soundex key, and can thus be used to simplify searches in databases where you know the pronunciation but not the spelling. This soundex function returns a string 4 characters long, starting with a letter.

This particular soundex function is one described by Donald Knuth in "The Art Of Computer Programming, vol. 3: Sorting And Searching", Addison-Wesley (1973), pp. 391-392.

Example 1. Soundex Examples

```
soundex ("Euler") == soundex ("Ellery") == 'E460';
soundex ("Gauss") == soundex ("Ghosh") == 'G200';
```

```
soundex ("Hilbert") == soundex ("Heilbronn") == 'H416';
soundex ("Knuth") == soundex ("Kant") == 'K530';
soundex ("Lloyd") == soundex ("Ladd") == 'L300';
soundex ("Lukasiewicz") == soundex ("Lissajous") == 'L222';
```

sprintf (PHP 3, PHP 4)

Return a formatted string

```
string sprintf (string format [, mixed args...])
```

Returns a string produced according to the formatting string *format*.

The format string is composed of zero or more directives: ordinary characters (excluding %) that are copied directly to the result, and *conversion specifications*, each of which results in fetching its own parameter. This applies to both **sprintf()** and **printf()**.

Each conversion specification consists of a percent sign (%), followed by one or more of these elements, in order:

1. An optional *padding specifier* that says what character will be used for padding the results to the right string size. This may be a space character or a 0 (zero character). The default is to pad with spaces. An alternate padding character can be specified by prefixing it with a single quote ('). See the examples below.
2. An optional *alignment specifier* that says if the result should be left-justified or right-justified. The default is right-justified; a - character here will make it left-justified.
3. An optional number, a *width specifier* that says how many characters (minimum) this conversion should result in.
4. An optional *precision specifier* that says how many decimal digits should be displayed for floating-point numbers. This option has no effect for other types than double. (Another function useful for formatting numbers is **number_format()**.)
5. A *type specifier* that says what type the argument data should be treated as. Possible types:

```
% - a literal percent character. No argument is required.  
b - the argument is treated as an integer, and presented as a binary number.  
c - the argument is treated as an integer, and presented as the character with that ASCII value.  
d - the argument is treated as an integer, and presented as a decimal number.  
f - the argument is treated as a double, and presented as a floating-point number.  
o - the argument is treated as an integer, and presented as an octal number.  
s - the argument is treated as and presented as a string.  
x - the argument is treated as an integer and presented as a hexadecimal number (with lowercase letters).  
X - the argument is treated as an integer and presented as a hexadecimal number (with uppercase letters).
```

See also: **printf()**, **sscanf()**, **fscanf()**, and **number_format()**.

Example 1. **Sprintf(): zero-padded integers**

```
$isodate = sprintf ("%04d-%02d-%02d", $year, $month, $day);
```

Example 2. sprintf(): formatting currency

```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money will output "123.1";
$formatted = sprintf ("%01.2f", $money);
// echo $formatted will output "123.10"
```

strncasecmp (PHP 4 >= 4.0.2)

Binary safe case-insensitive string comparison of the first n characters

```
int strncasecmp (string str1, string str2, int len)
```

This function is similar to **strcasecmp()**, with the difference that you can specify the (upper limit of the) number of characters (*len*) from each string to be used in the comparison. If any of the strings is shorter than *len*, then the length of that string will be used for the comparison.

Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

See also **ereg()**, **strcasecmp()**, **strcmp()**, **substr()**, **stristr()**, and **strrstr()**.

strcasecmp (PHP 3>= 3.0.2, PHP 4)

Binary safe case-insensitive string comparison

```
int strcasecmp (string str1, string str2)
```

Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

Example 1. strcasecmp() example

```
$var1 = "Hello";
$var2 = "hello";
if (!strcasecmp ($var1, $var2)) {
    echo '$var1 is equal to $var2 in a case-insensitive string comparison';
}
```

See also **ereg()**, **strcmp()**, **substr()**, **stristr()**, **strncasecmp()**, and **strrstr()**.

strchr (PHP 3, PHP 4)

Find the first occurrence of a character

```
string strchr (string haystack, string needle)
```

This function is an alias for **strrstr()**, and is identical in every way.

strcmp (PHP 3, PHP 4)

Binary safe string comparison

```
int strcmp (string str1, string str2)
```

Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

Note that this comparison is case sensitive.

See also **ereg()**, **strcasecmp()**, **substr()**, **stristr()**, **strncasecmp()**, **strncmp()**, and **strrstr()**.

strcoll (unknown)

Locale based string comparison

```
int strcoll (string str1, string str2)
```

Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal. **strcoll()** uses the current locale for doing the comparisons. If the current locale is C or POSIX, this function is equivalent to **strcmp()**.

Note that this comparison is case sensitive, and unlike **strcmp()** this function is not binary safe.

Note: Added in PHP 4.0.5.

See also **ereg()**, **strcmp()**, **strcasecmp()**, **substr()**, **stristr()**, **strncasecmp()**, **strncmp()**, **strrstr()**, and **setlocale()**.

strcspn (PHP 3>= 3.0.3, PHP 4)

Find length of initial segment not matching mask

```
int strcspn (string str1, string str2)
```

Returns the length of the initial segment of *str1* which does *not* contain any of the characters in *str2*.

See also **strspn()**.

strip_tags (PHP 3>= 3.0.8, PHP 4 >= 4.0b2)

Strip HTML and PHP tags from a string

```
string strip_tags (string str [, string allowable_tags])
```

This function tries to strip all HTML and PHP tags from the given string. It errors on the side of caution in case of incomplete or bogus tags. It uses the same tag stripping state machine as the **fgetss()** function.

You can use the optional second parameter to specify tags which should not be stripped.

Note: *Allowable_tags* was added in PHP 3.0.13, PHP4B3.

stripcslashes (PHP 4 >= 4.0b4)

Un-quote string quoted with **addcslashes()**

```
string stripcslashes (string str)
```

Returns a string with backslashes stripped off. Recognizes C-like \n, \r ..., octal and hexadecimal representation.

Note: Added in PHP4b3-dev.

See also **addcslashes()**.

stripslashes (PHP 3, PHP 4)

Un-quote string quoted with **addslashes()**

```
string stripslashes (string str)
```

Returns a string with backslashes stripped off. (\` becomes ' and so on.) Double backslashes are made into a single backslash.

See also **addslashes()**.

stristr (PHP 3>= 3.0.6, PHP 4)

Case-insensitive **strstr()**

```
string stristr (string haystack, string needle)
```

Returns all of *haystack* from the first occurrence of *needle* to the end. *needle* and *haystack* are examined in a case-insensitive manner.

If *needle* is not found, returns false.

If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

See also **strchr()**, **strrchr()**, **substr()**, and **ereg()**.

strlen (PHP 3, PHP 4)

Get string length

```
int strlen (string str)
```

Returns the length of *string*.

strnatcmp (PHP 4 >= 4.0RC2)

String comparisons using a "natural order" algorithm

```
int strnatcmp (string str1, string str2)
```

This function implements a comparison algorithm that orders alphanumeric strings in the way a human being would, this is described as a "natural ordering". An example of the difference between this algorithm and the regular computer string sorting algorithms (used in **strcmp()**) can be seen below:

```
$arr1 = $arr2 = array ("img12.png", "img10.png", "img2.png", "img1.png");
echo "Standard string comparison\n";
usort($arr1,"strcmp");
print_r($arr1);
echo "\nNatural order string comparison\n";
usort($arr2,"strnatcmp");
print_r($arr2);
```

The code above will generate the following output:

```
Standard string comparison
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
)

Natural order string comparison
Array
(
    [0] => img1.png
    [1] => img2.png
    [2] => img10.png
    [3] => img12.png
)
```

For more information see: Martin Pool's Natural Order String Comparison (<http://www.linuxcare.com.au/projects/natsort/>) page.

Similar to other string comparison functions, this one returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

Note that this comparison is case sensitive.

See also **ereg()**, **strcasecmp()**, **substr()**, **stristr()**, **strcmp()**, **strncmp()**, **strncasecmp()**, **strnatcasecmp()**, **strrstr()**, **natsort()** and **natcasesort()**.

strnatcasecmp (PHP 4 >= 4.0RC2)

Case insensitive string comparisons using a "natural order" algorithm

```
int strnatcasecmp (string str1, string str2)
```

This function implements a comparison algorithm that orders alphanumeric strings in the way a human being would. The behavior of this function is similar to **strnatcmp()**, except that the comparison is not case sensitive. For more information see: Martin Pool's Natural Order String Comparison (<http://www.linuxcare.com.au/projects/natsort/>) page.

Similar to other string comparison functions, this one returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

See also **ereg()**, **strcasecmp()**, **substr()**, **stristr()**, **strcmp()**, **strncmp()**, **strncasecmp()**, **strnatcmp()**, and **strrstr()**.

strncmp (PHP 4 >= 4.0b4)

Binary safe string comparison of the first n characters

```
int strncmp (string str1, string str2, int len)
```

This function is similar to **strcmp()**, with the difference that you can specify the (upper limit of the) number of characters (*len*) from each string to be used in the comparison. If any of the strings is shorter than *len*, then the length of that string will be used for the comparison.

Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

Note that this comparison is case sensitive.

See also **ereg()**, **strcasecmp()**, **strncasecmp()**, **substr()**, **stristr()**, **strcmp()**, and **strrstr()**.

str_pad (PHP 4 >= 4.0.1)

Pad a string to a certain length with another string

```
string str_pad (string input, int pad_length [, string pad_string [, int pad_type]])
```

This function pads the *input* string on the left, the right, or both sides to the specified padding length. If the optional argument *pad_string* is not supplied, the *input* is padded with spaces, otherwise it is padded with characters from *pad_string* up to the limit.

Optional argument *pad_type* can be **STR_PAD_RIGHT**, **STR_PAD_LEFT**, or **STR_PAD_BOTH**. If *pad_type* is not specified it is assumed to be **STR_PAD_RIGHT**.

If the value of *pad_length* is negative or less than the length of the input string, no padding takes place.

Example 1. str_pad() example

```
$input = "Alien";
print str_pad($input, 10);                                // produces "Alien      "
print str_pad($input, 10, "-=", STR_PAD_LEFT);    // produces "-====Alien"
print str_pad($input, 10, "_", STR_PAD_BOTH);   // produces "__Alien__"
```

strpos (PHP 3, PHP 4)

Find position of first occurrence of a string

```
int strpos (string haystack, string needle [, int offset])
```

Returns the numeric position of the first occurrence of *needle* in the *haystack* string. Unlike the **strrpos()**, this function can take a full string as the *needle* parameter and the entire string will be used.

If *needle* is not found, returns false.

Note: It is easy to mistake the return values for "character found at position 0" and "character not found". Here's how to detect the difference:

```
// in PHP 4.0b3 and newer:  
$pos = strpos ($mystring, "b");  
if ($pos === false) { // note: three equal signs  
    // not found...  
}  
  
// in versions older than 4.0b3:  
$pos = strpos ($mystring, "b");  
if (is_string ($pos) && !$pos) {  
    // not found...  
}
```

If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

The optional *offset* parameter allows you to specify which character in *haystack* to start searching. The position returned is still relative to the the beginning of *haystack*.

See also **strrpos()**, **strrchr()**, **substr()**, **stristr()**, and **strrstr()**.

strrchr (PHP 3, PHP 4)

Find the last occurrence of a character in a string

```
string strrchr (string haystack, string needle)
```

This function returns the portion of *haystack* which starts at the last occurrence of *needle* and goes until the end of *haystack*.

Returns false if *needle* is not found.

If *needle* contains more than one character, the first is used.

If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Example 1. Strrchr() example

```
// get last directory in $PATH  
$dir = substr ( strrchr ($PATH, ":"), 1 );  
  
// get everything after last newline  
$text = "Line 1\nLine 2\nLine 3";  
$last = substr ( strrchr ($text, 10), 1 );
```

See also **substr()**, **stristr()**, and **strrstr()**.

str_repeat (PHP 4 >= 4.0b4)

Repeat a string

```
string str_repeat (string input, int multiplier)
```

Returns *input*_str repeated *multiplier* times. *multiplier* has to be greater than 0.

Example 1. Str_repeat() example

```
echo str_repeat ("-=", 10);
```

This will output "-=-=-=-=-=-=-".

Note: This function was added in PHP 4.0.

strrev (PHP 3, PHP 4)

Reverse a string

```
string strrev (string string)
```

Returns *string*, reversed.

strrpos (PHP 3, PHP 4)

Find position of last occurrence of a char in a string

```
int strrpos (string haystack, char needle)
```

Returns the numeric position of the last occurrence of *needle* in the *haystack* string. Note that the needle in this case can only be a single character. If a string is passed as the needle, then only the first character of that string will be used.

If *needle* is not found, returns false.

Note: It is easy to mistake the return values for "character found at position 0" and "character not found". Here's how to detect the difference:

```
// in PHP 4.0b3 and newer:  
$pos = strrpos ($mystring, "b");  
if ($pos === false) { // note: three equal signs  
    // not found...  
}  
  
// in versions older than 4.0b3:  
$pos = strrpos ($mystring, "b");  
if (is_string ($pos) && !$pos) {  
    // not found...  
}
```

If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.
 See also **strpos()**, **strrchr()**, **substr()**, **stristr()**, and **strstr()**.

strspn (PHP 3>= 3.0.3, PHP 4)

Find length of initial segment matching mask

```
int strspn (string str1, string str2)
```

Returns the length of the initial segment of *str1* which consists entirely of characters in *str2*.

```
strspn ("42 is the answer, what is the question ...", "1234567890");
```

will return 2 as result.

See also **strcspn()**.

strstr (PHP 3, PHP 4)

Find first occurrence of a string

```
string strstr (string haystack, string needle)
```

Returns all of *haystack* from the first occurrence of *needle* to the end.

If *needle* is not found, returns false.

If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Note: Note that this function is case-sensitive. For case-insensitive searches, use **stristr()**.

Example 1. Strstr() example

```
$email = 'sterling@designmultimedia.com';
$domain = strstr ($email, '@');
print $domain; // prints @designmultimedia.com
```

See also **stristr()**, **strrchr()**, **substr()**, and **ereg()**.

strtok (PHP 3, PHP 4)

Tokenize string

```
string strtok (string arg1, string arg2)
```

strtok() is used to tokenize a string. That is, if you have a string like "This is an example string" you could tokenize this string into its individual words by using the space character as the token.

Example 1. Strtok() example

```
$string = "This is an example string";
$tok = strtok ($string, " ");
while ($tok) {
    echo "Word=$tok<br>";
    $tok = strtok (" ");
}
```

Note that only the first call to strtok uses the string argument. Every subsequent call to strtok only needs the token to use, as it keeps track of where it is in the current string. To start over, or to tokenize a new string you simply call strtok with the string argument again to initialize it. Note that you may put multiple tokens in the token parameter. The string will be tokenized when any one of the characters in the argument are found.

Also be careful that your tokens may be equal to "0". This evaluates to false in conditional expressions.

See also **split()** and **explode()**.

strtolower (PHP 3, PHP 4)

Make a string lowercase

```
string strtolower (string str)
```

Returns *string* with all alphabetic characters converted to lowercase.

Note that 'alphabetic' is determined by the current locale. This means that in i.e. the default "C" locale, characters such as umlaut-A (Ä) will not be converted.

Example 1. Strtolower() example

```
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtolower($str);
print $str; # Prints mary had a little lamb and she loved it so
```

See also **strtoupper()** and **ucfirst()**.

strtoupper (PHP 3, PHP 4)

Make a string uppercase

```
string strtoupper (string string)
```

Returns *string* with all alphabetic characters converted to uppercase.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

Example 1. Strtoupper() example

```
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtoupper ($str);
print $str; # Prints MARY HAD A LITTLE LAMB AND SHE LOVED IT SO
```

See also **strtolower()** and **ucfirst()**.

str_replace (PHP 3>= 3.0.6, PHP 4)

Replace all occurrences of the search string in *subject* with the replacement string

```
mixed str_replace (mixed search, mixed replace, mixed subject)
```

This function replaces all occurrences of *search* in *subject* with the given *replace* value. If you don't need fancy replacing rules, you should always use this function instead of **ereg_replace()** or **preg_replace()**.

In PHP 4.0.5 and later, every parameter to **str_replace()** can be an array.

If *subject* is an array, then the search and replace is performed on every entry of *subject*, and the return value is an array as well.

If *search* and *replace* are arrays, then **str_replace()** takes a value from each array and uses them to do search and replace on *subject*. If *replace* has fewer values than *search*, then an empty string is used for the rest of replacement values. If *search* is an array and *replace* is a string; then this replacement string is used for every value of *search*. The converse would not make sense, though.

Example 1. str_replace() example

```
$bodytag = str_replace ("%body%", "black", "<body text=%body%>");
```

This function is binary safe.

Note: **str_replace()** was added in PHP 3.0.6, but was buggy up until PHP 3.0.8.

See also **ereg_replace()**, **preg_replace()**, and **strtr()**.

strtr (PHP 3, PHP 4)

Translate certain characters

```
string strtr (string str, string from, string to)
```

This function operates on *str*, translating all occurrences of each character in *from* to the corresponding character in *to* and returning the result.

If *from* and *to* are different lengths, the extra characters in the longer of the two are ignored.

Example 1. Strtr() example

```
$addr = strtr($addr, "ääö", "ao");
```

strtr() can be called with only two arguments. If called with two arguments it behaves in a new way: *from* then has to be an array that contains string -> string pairs that will be replaced in the source string. **strtr()** will always look for the longest possible match first and will *NOT* try to replace stuff that it has already worked on.

Examples:

```
$trans = array ("hello" => "hi", "hi" => "hello");
echo strtr("hi all, I said hello", $trans) . "\n";
```

This will show: "hello all, I said hi",

Note: This feature (two arguments) was added in PHP 4.0.

See also **ereg_replace()**.

substr (PHP 3, PHP 4)

Return part of a string

```
string substr (string string, int start [, int length])
```

Substr returns the portion of *string* specified by the *start* and *length* parameters.

If *start* is positive, the returned string will start at the *start*'th position in *string*, counting from zero. For instance, in the string 'abcdef', the character at position 0 is 'a', the character at position 2 is 'c', and so forth.

Examples:

```
$rest = substr ("abcdef", 1);      // returns "bcdef"
$rest = substr ("abcdef", 1, 3); // returns "bcd"
```

If *start* is negative, the returned string will start at the *start*'th character from the end of *string*.

Examples:

```
$rest = substr ("abcdef", -1);     // returns "f"
$rest = substr ("abcdef", -2);    // returns "ef"
$rest = substr ("abcdef", -3, 1); // returns "d"
```

If *length* is given and is positive, the string returned will end *length* characters from *start*. If this would result in a string with negative length (because the start is past the end of the string), then the returned string will contain the single character at *start*.

If *length* is given and is negative, the string returned will end *length* characters from the end of *string*. If this would result in a string with negative length, then the returned string will contain the single character at *start*.

Examples:

```
$rest = substr ("abcdef", 1, -1); // returns "bcde"
```

See also **strrchr()** and **ereg()**.

substr_count (PHP 4 >= 4.0RC2)

Count the number of substring occurrences

```
int substr_count (string haystack, string needle)
```

substr_count() returns the number of times the *needle* substring occurs in the *haystack* string.

Example 1. substr_count() example

```
print substr_count("This is a test", "is"); // prints out 2
```

substr_replace (PHP 4 >= 4.0b4)

Replace text within a portion of a string

```
string substr_replace (string string, string replacement, int start [, int length])
```

substr_replace() replaces the part of *string* delimited by the *start* and (optionally) *length* parameters with the string given in *replacement*. The result is returned.

If *start* is positive, the replacing will begin at the *start*'th offset into *string*.

If *start* is negative, the replacing will begin at the *start*'th character from the end of *string*.

If *length* is given and is positive, it represents the length of the portion of *string* which is to be replaced. If it is negative, it represents the number of characters from the end of *string* at which to stop replacing. If it is not given, then it will default to *strlen(string)*; i.e. end the replacing at the end of *string*.

Example 1. Substr_replace() example

```
<?php
$var = 'ABCDEFGH:/MNRPQR/';
echo "Original: $var<br>\n";

/* These two examples replace all of $var with 'bob'. */
echo substr_replace ($var, 'bob', 0) . "<br>\n";
echo substr_replace ($var, 'bob', 0, strlen ($var)) . "<br>\n";

/* Insert 'bob' right at the beginning of $var. */
echo substr_replace ($var, 'bob', 0, 0) . "<br>\n";

/* These next two replace 'MNRPQR' in $var with 'bob'. */
echo substr_replace ($var, 'bob', 10, -1) . "<br>\n";
echo substr_replace ($var, 'bob', -7, -1) . "<br>\n";
```

```
/* Delete 'MNRPQR' from $var. */
echo substr_replace ($var, "", 10, -1) . "<br>\n";
?>
```

See also **[str_replace\(\)](#)** and **[substr\(\)](#)**.

Note: **Substr_replace()** was added in PHP 4.0.

trim (PHP 3, PHP 4)

Strip whitespace from the beginning and end of a string

```
string trim (string str)
```

This function strips whitespace from the start and the end of a string and returns the stripped string. The whitespace characters it currently strips are: "\n", "\r", "\t", "\v", "\0", and a plain space.

See also **[chop\(\)](#)**, **[rtrim\(\)](#)** and **[ltrim\(\)](#)**.

ucfirst (PHP 3, PHP 4)

Make a string's first character uppercase

```
string ucfirst (string str)
```

Capitalizes the first character of *str* if that character is alphabetic.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

Example 1. Ucfirst() example

```
$text = 'mary had a little lamb and she loved it so.';
$text = ucfirst ($text); // $text is now Mary had a little lamb
                           // and she loved it so.
```

See also **[strtoupper\(\)](#)** and **[strtolower\(\)](#)**.

ucwords (PHP 3>= 3.0.3, PHP 4)

Uppercase the first character of each word in a string

```
string ucwords (string str)
```

Capitalizes the first character of each word in *str* if that character is alphabetic.

Example 1. ucwords() example

```
$text = "mary had a little lamb and she loved it so.";
$text = ucwords($text); // $text is now: Mary Had A Little
// Lamb And She Loved It So.
```

Note: The definition of a word is any string of characters that is immediately after a whitespace (These are: space, form-feed, newline, carriage return, horizontal tab, and vertical tab).

See also **strtoupper()**, **strtolower()** and **ucfirst()**.

wordwrap (PHP 4 >= 4.0.2)

Wraps a string to a given number of characters using a string break character.

```
string wordwrap (string str [, int width [, string break [, int cut] ]])
```

Wraps the string *str* at the column number specified by the (optional) *width* parameter. The line is broken using the (optional) *break* parameter.

wordwrap() will automatically wrap at column 75 and break using '\n' (newline) if *width* or *break* are not given.

If the *cut* is set to 1, the string is always wrapped at the specified width. So if you have a word that is larger than the given width, it is broken appart. (See second example).

Note: The *cut* parameter was added in PHP 4.0.3.

Example 1. wordwrap() example

```
$text = "The quick brown fox jumped over the lazy dog.";
$newtext = wordwrap( $text, 20 );

echo "$newtext\n";
```

This example would display:

```
The quick brown fox
jumped over the lazy dog.
```

Example 2. wordwrap() example

```
$text = "A very long wooooooooooooord.";
$newtext = wordwrap( $text, 8, "\n", 1);

echo "$newtext\n";
```

This example would display:

```
A very  
long  
wooooooo  
ooooord.
```

See also **nl2br()**.

LXXI. Sybase functions

sybase_affected_rows (PHP 3>= 3.0.6, PHP 4)

get number of affected rows in last query

```
int sybase_affected_rows ([int link_identifier])
```

Returns: The number of affected rows by the last query.

sybase_affected_rows() returns the number of rows affected by the last INSERT, UPDATE or DELETE query on the server associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

This command is not effective for SELECT statements, only on statements which modify records. To retrieve the number of rows returned from a SELECT, use **sybase_num_rows()**.

Note: This function is only available using the CT library interface to Sybase, and not the DB library.

sybase_close (PHP 3, PHP 4)

close Sybase connection

```
bool sybase_close (int link_identifier)
```

Returns: true on success, false on error

sybase_close() closes the link to a Sybase 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.

sybase_close() will not close persistent links generated by **sybase_pconnect()**.

See also: **sybase_connect()**, **sybase_pconnect()**.

sybase_connect (PHP 3, PHP 4)

open Sybase server connection

```
int sybase_connect (string servername, string username, string password [, string charset])
```

Returns: A positive Sybase link identifier on success, or false on error.

sybase_connect() establishes a connection to a Sybase server. The *servername* argument has to be a valid *servername* that is defined in the 'interfaces' file.

In case a second call is made to **sybase_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 **sybase_close()**.

See also **sybase_pconnect()**, **sybase_close()**.

sybase_data_seek (PHP 3, PHP 4)

move internal row pointer

```
bool sybase_data_seek (int result_identifier, int row_number)
```

Returns: true on success, false on failure

`sybase_data_seek()` moves the internal row pointer of the Sybase result associated with the specified result identifier to pointer to the specified row number. The next call to `sybase_fetch_row()` would return that row.

See also: `sybase_data_seek()`.

sybase_fetch_array (PHP 3, PHP 4)

fetch row as array

```
array sybase_fetch_array (int result)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

`sybase_fetch_array()` is an extended version of `sybase_fetch_row()`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

An important thing to note is that using `sybase_fetch_array()` is NOT significantly slower than using `sybase_fetch_row()`, while it provides a significant added value.

For further details, also see `sybase_fetch_row()`

sybase_fetch_field (PHP 3, PHP 4)

get field information

```
object sybase_fetch_field (int result [, int field_offset])
```

Returns an object containing field information.

`sybase_fetch_field()` can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retrieved by `sybase_fetch_field()` is retrieved.

The properties of the object are:

- name - column name. if the column is a result of a function, this property is set to computed#N, where #N is a serial number.
- column_source - the table from which the column was taken
- max_length - maximum length of the column
- numeric - 1 if the column is numeric
- type - datatype of the column

See also `sybase_field_seek()`

sybase_fetch_object (PHP 3, PHP 4)

fetch row as object

```
int sybase_fetch_object (int result)
```

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

sybase_fetch_object() is similar to **sybase_fetch_array()**, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

Speed-wise, the function is identical to **sybase_fetch_array()**, and almost as quick as **sybase_fetch_row()** (the difference is insignificant).

See also: **sybase_fetch-array()** and **sybase_fetch-row()**.

sybase_fetch_row (PHP 3, PHP 4)

get row as enumerated array

```
array sybase_fetch_row (int result)
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

sybase_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 **sybase_fetch_rows()** would return the next row in the result set, or false if there are no more rows.

See also: **sybase_fetch_array()**, **sybase_fetch_object()**, **sybase_data_seek()**, **sybase_fetch_lengths()**, and **sybase_result()**.

sybase_field_seek (PHP 3, PHP 4)

set field offset

```
int sybase_field_seek (int result, int field_offset)
```

Seeks to the specified field offset. If the next call to **sybase_fetch_field()** won't include a field offset, this field would be returned.

See also: **sybase_fetch_field()**.

sybase_free_result (PHP 3, PHP 4)

free result memory

```
bool sybase_free_result (int result)
```

sybase_free_result() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script ends. You may call **sybase_free_result()** with the result identifier as an argument and the associated result memory will be freed.

sybase_get_last_message (PHP 3, PHP 4)

Returns the last message from the server

```
string sybase_get_last_message (void )
```

sybase_get_last_message() returns the last message reported by the server.

sybase_min_client_severity (PHP 3, PHP 4)

Sets minimum client severity

```
void sybase_min_client_severity (int severity)
```

sybase_min_client_severity() sets the minimum client severity level.

Note: This function is only available using the CT library interface to Sybase, and not the DB library.

See also: **sybase_min_server_severity()**.

sybase_min_error_severity (PHP 3, PHP 4)

Sets minimum error severity

```
void sybase_min_error_severity (int severity)
```

sybase_min_error_severity() sets the minimum error severity level.

See also: **sybase_min_message_severity()**.

sybase_min_message_severity (PHP 3, PHP 4)

Sets minimum message severity

```
void sybase_min_message_severity (int severity)
```

sybase_min_message_severity() sets the minimum message severity level.

See also: **sybase_min_error_severity()**.

sybase_min_server_severity (PHP 3, PHP 4)

Sets minimum server severity

```
void sybase_min_server_severity (int severity)
```

sybase_min_server_severity() sets the minimum server severity level.

Note: This function is only available using the CT library interface to Sybase, and not the DB library.

See also: [sybase_min_client_severity\(\)](#).

sybase_num_fields (PHP 3, PHP 4)

get number of fields in result

```
int sybase_num_fields (int result)
```

sybase_num_fields() returns the number of fields in a result set.

See also: [sybase_db_query\(\)](#), [sybase_query\(\)](#), [sybase_fetch_field\(\)](#), [sybase_num_rows\(\)](#).

sybase_num_rows (PHP 3, PHP 4)

get number of rows in result

```
int sybase_num_rows (int result)
```

sybase_num_rows() returns the number of rows in a result set.

See also: [sybase_db_query\(\)](#), [sybase_query\(\)](#) and, [sybase_fetch_row\(\)](#).

sybase_pconnect (PHP 3, PHP 4)

open persistent Sybase connection

```
int sybase_pconnect (string servername, string username, string password [, string charset])
```

Returns: A positive Sybase persistent link identifier on success, or false on error

sybase_pconnect() acts very much like **sybase_connect()** with two major differences.

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 (**sybase_close()** will not close links established by **sybase_pconnect()**).

This type of links is therefore called 'persistent'.

sybase_query (PHP 3, PHP 4)

send Sybase query

```
int sybase_query (string query, int link_identifier)
```

Returns: A positive Sybase result identifier on success, or false on error.

`sybase_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 `sybase_connect()` was called, and use it.

See also: `sybase_db_query()`, `sybase_select_db()`, and `sybase_connect()`.

sybase_result (PHP 3, PHP 4)

get result data

```
string sybase_result (int result, int row, mixed field)
```

Returns: The contents of the cell at the row and offset in the specified Sybase result set.

`sybase_result()` returns the contents of one cell from a Sybase result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (tablename.fieldname). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than `sybase_result()`. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: `sybase_fetch_row()`, `sybase_fetch_array()`, and `sybase_fetch_object()`.

sybase_select_db (PHP 3, PHP 4)

select Sybase database

```
bool sybase_select_db (string database_name, int link_identifier)
```

Returns: true on success, false on error

`sybase_select_db()` sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if `sybase_connect()` was called, and use it.

Every subsequent call to `sybase_query()` will be made on the active database.

See also: `sybase_connect()`, `sybase_pconnect()`, and `sybase_query()`

LXXII. URL Functions

base64_decode (PHP 3, PHP 4)

Decodes data encoded with MIME base64

```
string base64_decode (string encoded_data)
```

Base64_decode() decodes *encoded_data* and returns the original data. The returned data may be binary.

See also: **base64_encode()**, RFC-2045 section 6.8.

base64_encode (PHP 3, PHP 4)

Encodes data with MIME base64

```
string base64_encode (string data)
```

Base64_encode() returns *data* encoded with base64. This encoding is designed to make binary data survive transport through transport layers that are not 8-bit clean, such as mail bodies.

Base64-encoded data takes about 33% more space than the original data.

See also: **base64_decode()**, **chunk_split()**, RFC-2045 section 6.8.

parse_url (PHP 3, PHP 4)

Parse a URL and return its components

```
array parse_url (string url)
```

This function returns an associative array returning any of the various components of the URL that are present. This includes the "scheme", "host", "port", "user", "pass", "path", "query", and "fragment".

rawurlencode (PHP 3, PHP 4)

Decode URL-encoded strings

```
string rawurlencode (string str)
```

Returns a string in which the sequences with percent (%) signs followed by two hex digits have been replaced with literal characters. For example, the string

foo%20bar%40baz

decodes into

```
foo  
bar@baz
```

See also **rawurldecode()**, **urldecode()**, **urlencode()**.

rawurlencode (PHP 3, PHP 4)

URL-encode according to RFC1738

```
string rawurlencode (string str)
```

Returns a string in which all non-alphanumeric characters except

-_.-

have been replaced with a percent (%) sign followed by two hex digits. This is the encoding described in RFC1738 for protecting literal characters from being interpreted as special URL delimiters, and for protecting URL's from being mangled by transmission media with character conversions (like some email systems). For example, if you want to include a password in an ftp url:

Example 1. Rawurlencode() example 1

```
echo '<A HREF="ftp://user:', rawurlencode ('foo @+%/'),
      '@ftp.my.com/x.txt">;
```

Or, if you pass information in a path info component of the url:

Example 2. Rawurlencode() example 2

```
echo '<A HREF="http://x.com/department_list_script/',
      rawurlencode ('sales and marketing/Miami'), '">;
```

See also **rawurldecode()**, **urldecode()**, **urlencode()**.

urlencode (PHP 3, PHP 4)

Decodes URL-encoded string

```
string urlencode (string str)
```

Decodes any %## encoding in the given string. The decoded string is returned.

Example 1. Urldecode() example

```
$a = split ('&', $querystring);
$i = 0;
while ($i < count ($a)) {
    $b = split ('=', $a [$i]);
    echo 'Value for parameter ', htmlspecialchars (urldecode ($b [0])),
         ' is ', htmlspecialchars (urldecode ($b [1])), "<BR>";
    $i++;
}
```

See also **urlencode()**, **rawurlencode()**, **rawurldecode()**.

urlencode (PHP 3, PHP 4)

URL-encodes string

```
string urlencode (string str)
```

Returns a string in which all non-alphanumeric characters except -_. have been replaced with a percent (%) sign followed by two hex digits and spaces encoded as plus (+) signs. It is encoded the same way that the posted data from a WWW form is encoded, that is the same way as in `application/x-www-form-urlencoded` media type. This differs from the RFC1738 encoding (see **rawurlencode()**) in that for historical reasons, spaces are encoded as plus (+) signs. This function is convenient when encoding a string to be used in a query part of an URL, as a convenient way to pass variables to the next page:

Example 1. Urlencode() example

```
echo '<A HREF="mycgi?foo=' . urlencode ($userinput) . '">' ;
```

Note: Be careful about variables that may match HTML entities. Things like &, © and £ are parsed by the browser and the actual entity is used instead of the desired variable name. This is an obvious hassle that the W3C has been telling people about for years. The reference is here: <http://www.w3.org/TR/html4/appendix/notes.html#h-B.2.2> PHP supports changing the argument separator to the W3C-suggested semi-colon through the `arg_separator`.ini directive. Unfortunately most user agents do not send form data in this semi-colon separated format. A more portable way around this is to use & instead of & as the separator. You don't need to change PHP's `arg_separator` for this. Leave it as &, but simply encode your URLs using **htmlentities()**(`urlencode($data)`).

Example 2. Urlencode/htmlentities() example

```
echo '<A HREF="mycgi?foo=' . htmlentities (urlencode ($userinput)) . '">' ;
```

See also **urldecode()**, **htmlentities()**, **rawurldecode()**, **rawurlencode()**.

LXXIII. Variable Functions

doubleval (PHP 3, PHP 4)

Get double value of a variable

```
double doubleval (mixed var)
```

Returns the double (floating point) value of *var*.

Var may be any scalar type. You cannot use **doubleval()** on arrays or objects.

```
$var = '122.34343The';
$double_value_of_var = doubleval ($var);
print $double_value_of_var; // prints 122.34343
```

See also **intval()**, **strval()**, **settype()** and [Type juggling](#).

empty (unknown)

Determine whether a variable is set

```
int empty (mixed var)
```

Returns false if *var* is set and has a non-empty or non-zero value; true otherwise.

```
$var = 0;

if (empty($var)) { // evaluates true
    echo '$var is either 0 or not set at all';
}

if (!isset($var)) { // evaluates false
    echo '$var is not set at all';
}
```

Note that this is meaningless when used on anything which isn't a variable; i.e. **empty (addslashes (\$name))** has no meaning since it would be checking whether something which isn't a variable is a variable with a false value.

See also **isset()** and **unset()**.

gettype (PHP 3, PHP 4)

Get the type of a variable

```
string gettype (mixed var)
```

Returns the type of the PHP variable *var*.

Possible values for the returned string are:

- "boolean"

- "integer"
- "double"
- "string"
- "array"
- "object"
- "resource"
- "user function" (PHP 3 only, deprecated)
- "unknown type"

For PHP 4, you should use **function_exists()** and **method_exists()** to replace the prior usage of **gettype()** on a function.

See also **settype()**.

intval (PHP 3, PHP 4)

Get integer value of a variable

```
int intval (mixed var [, int base])
```

Returns the integer value of *var*, using the specified base for the conversion (the default is base 10).

Var may be any scalar type. You cannot use **intval()** on arrays or objects.

See also **doubleval()**, **strval()**, **settype()** and [Type juggling](#).

is_array (PHP 3, PHP 4)

Finds whether a variable is an array

```
bool is_array (mixed var)
```

Returns true if *var* is an array, false otherwise.

See also **is_double()**, **is_float()**, **is_int()**, **is_integer()**, **is_real()**, **is_string()**, **is_long()**, and **is_object()**.

is_bool (PHP 4 >= 4.0b4)

Finds out whether a variable is a boolean

```
bool is_bool (mixed var)
```

Returns true if the *var* parameter is a boolean.

See also **is_array()**, **is_double()**, **is_float()**, **is_int()**, **is_integer()**, **is_real()**, **is_string()**, **is_long()**, and **is_object()**.

is_double (PHP 3, PHP 4)

Finds whether a variable is a double

```
bool is_double (mixed var)
```

Returns true if *var* is a double, false otherwise.

See also **is_array()**, **is_bool()**, **is_float()**, **is_int()**, **is_integer()**, **is_real()**, **is_string()**, **is_long()**, and **is_object()**.

is_float (PHP 3, PHP 4)

Finds whether a variable is a float

```
bool is_float (mixed var)
```

This function is an alias for **is_double()**.

See also **is_double()**, **is_bool()**, **is_real()**, **is_int()**, **is_integer()**, **is_string()**, **is_object()**, **is_array()**, and **is_long()**.

is_int (PHP 3, PHP 4)

Find whether a variable is an integer

```
bool is_int (mixed var)
```

This function is an alias for **is_long()**.

See also **is_bool()**, **is_double()**, **is_float()**, **is_integer()**, **is_string()**, **is_real()**, **is_object()**, **is_array()**, and **is_long()**.

is_integer (PHP 3, PHP 4)

Find whether a variable is an integer

```
bool is_integer (mixed var)
```

This function is an alias for **is_long()**.

See also **is_bool()**, **is_double()**, **is_float()**, **is_int()**, **is_string()**, **is_real()**, **is_object()**, **is_array()**, and **is_long()**.

is_long (PHP 3, PHP 4)

Finds whether a variable is an integer

```
bool is_long (mixed var)
```

Returns true if *var* is an integer (long), false otherwise.

See also **is_bool()**, **is_double()**, **is_float()**, **is_int()**, **is_real()**, **is_string()**, **is_object()**, **is_array()**, and **is_integer()**.

is_numeric (PHP 4 >= 4.0RC1)

Finds whether a variable is a number or a numeric string

```
bool is_numeric (mixed var)
```

Returns true if *var* is a number or a numeric string, false otherwise.

See also **is_bool()**, **is_double()**, **is_float()**, **is_int()**, **is_real()**, **is_string()**, **is_object()**, **is_array()**, and **is_integer()**.

is_object (PHP 3, PHP 4)

Finds whether a variable is an object

```
bool is_object (mixed var)
```

Returns true if *var* is an object, false otherwise.

See also **is_bool()**, **is_long()**, **is_int()**, **is_integer()**, **is_float()**, **is_double()**, **is_real()**, **is_string()**, and **is_array()**.

is_real (PHP 3, PHP 4)

Finds whether a variable is a real

```
bool is_real (mixed var)
```

This function is an alias for **is_double()**.

See also **is_bool()**, **is_long()**, **is_int()**, **is_integer()**, **is_float()**, **is_double()**, **is_object()**, **is_string()**, and **is_array()**.

is_resource (PHP 4 >= 4.0b4)

Finds whether a variable is a resource

```
bool is_resource (mixed var)
```

is_resource() returns true if the variable given by the *var* parameter is a resource, otherwise it returns false.

Resources are things like file or database result handles that are allocated and freed by internal PHP functions and that may need some cleanup when they are no longer in use but haven't been freed by user code.

is_string (PHP 3, PHP 4)

Finds whether a variable is a string

```
bool is_string (mixed var)
```

Returns true if *var* is a string, false otherwise.

See also **is_bool()**, **is_long()**, **is_int()**, **is_integer()**, **is_float()**, **is_double()**, **is_real()**, **is_object()**, and **is_array()**.

isset (unknown)

Determine whether a variable is set

```
int isset (mixed var)
```

Returns true if *var* exists; false otherwise.

If a variable has been unset with **unset()**, it will no longer be **isset()**.

```
$a = "test";
echo isset ($a); // true
unset ($a);
echo isset ($a); // false
```

See also **empty()** and **unset()**.

print_r (PHP 4)

Prints human-readable information about a variable

```
void print_r (mixed expression)
```

This function displays information about the values of variables in a way that's readable by humans. If given a string, integer or double, the value itself will be printed. If given an array, values will be presented in a format that shows keys and elements. Similar notation is used for objects.

Compare **print_r()** to **var_dump()**.

```
<?php
$a = array (1, 2, array ("a", "b", "c"));
print_r ($a);
?>
```

Warning

This function will continue forever if given an array or object that contains a direct or indirect reference to itself or that contains an array or object on a deeper level that does so. This is especially true for **print_r(\$GLOBALS)**, as **\$GLOBALS** is itself a global variable and contains a reference to itself as such.

serialize (PHP 3>= 3.0.5, PHP 4)

Generates a storable representation of a value

```
string serialize (mixed value)
```

Serialize() returns a string containing a byte-stream representation of *value* that can be stored anywhere.

This is useful for storing or passing PHP values around without losing their type and structure.

To make the serialized string into a PHP value again, use **unserialize()**. **Serialize()** handles the types integer, double, string, array (multidimensional) and object (object properties will be serialized, but methods are lost).

Example 1. **Serialize()** example

```
// $session_data contains a multi-dimensional array with session
// information for the current user. We use serialize() to store
// it in a database at the end of the request.

$conn = odbc_connect ("webdb", "php", "chicken");
$stmt = odbc_prepare ($conn,
    "UPDATE sessions SET data = ? WHERE id = ?");
$sqldata = array (serialize($session_data), $PHP_AUTH_USER);
if (!odbc_execute ($stmt, &$sqldata)) {
    $stmt = odbc_prepare($conn,
        "INSERT INTO sessions (id, data) VALUES(?, ?)");
    if (!odbc_execute($stmt, &$sqldata)) {
        /* Something went wrong. Bitch, whine and moan. */
    }
}
```

settype (PHP 3, PHP 4)

Set the type of a variable

```
int settype (string var, string type)
```

Set the type of variable *var* to *type*.

Possible values of *type* are:

- "integer"
- "double"
- "string"
- "array"
- "object"

Returns true if successful; otherwise returns false.

See also **gettype()**.

strval (PHP 3, PHP 4)

Get string value of a variable

```
string strval (mixed var)
```

Returns the string value of *var*.

var may be any scalar type. You cannot use **strval()** on arrays or objects.

See also **doubleval()**, **intval()**, **settype()** and [Type juggling](#).

unserialize (PHP 3>= 3.0.5, PHP 4)

Creates a PHP value from a stored representation

```
mixed unserialize (string str)
```

unserialize() takes a single serialized variable (see **serialize()**) and converts it back into a PHP value. The converted value is returned, and can be an integer, double, string, array or object. If an object was serialized, its methods are not preserved in the returned value.

Example 1. Unserialize() example

```
// Here, we use unserialize() to load session data from a database
// into $session_data. This example complements the one described
// with serialize().
$conn = odbc_connect ("webdb", "php", "chicken");
$stmt = odbc_prepare ($conn, "SELECT data FROM sessions WHERE id = ?");
$sqldata = array ($PHP_AUTH_USER);
if (!odbc_execute ($stmt, &$sqldata) || !odbc_fetch_into ($stmt, &$tmp)) {
    // if the execute or fetch fails, initialize to empty array
    $session_data = array();
} else {
    // we should now have the serialized data in $tmp[0].
    $session_data = unserialize ($tmp[0]);
    if (!is_array ($session_data)) {
        // something went wrong, initialize to empty array
        $session_data = array();
    }
}
```

unset (unknown)

Unset a given variable

```
int unset (mixed var [, mixed var [, ...]])
```

unset() destroys the specified variables and returns true.

Example 1. Unset() example

```
// destroy a single variable
unset ($foo);
```

```
// destroy a single element of an array
unset ($bar['quux']);

// destroy more than one variable
unset ($foo1, $foo2, $foo3);
```

The behavior of **unset()** inside of a function can vary depending on what type of variable you are attempting to destroy.

If a globalized variable is **unset()** inside of a function, only the local variable is destroyed. The variable in the calling environment will retain the same value as before **unset()** was called.

```
function destroy_foo() {
    global $foo;
    unset($foo);
}

$foo = 'bar';
destroy_foo();
echo $foo;
```

The above example would output:

```
bar
```

If a variable that is PASSED BY REFERENCE is **unset()** inside of a function, only the local variable is destroyed. The variable in the calling environment will retain the same value as before **unset()** was called.

```
function foo(&$bar) {
    unset($bar);
    $bar = "blah";
}

$bar = 'something';
echo "$bar\n";

foo($bar);
echo "$bar\n";
```

The above example would output:

```
something
something
```

If a static variable is **unset()** inside of a function, **unset()** unsets the reference to the static variable, rather than the static variable itself.

```
function foo() {
    static $a;
    $a++;
    echo "$a\n";

    unset($a);
}
```

```
foo();
foo();
foo();
```

The above example would output:

```
1
2
3
```

If you would like to **unset()** a global variable inside of a function, you can use the **\$GLOBALS** array to do so:

```
function foo() {
    unset($GLOBALS['bar']);
}

$bar = "something";
foo();
```

See also **isset()** and **empty()**.

var_dump (PHP 3>= 3.0.5, PHP 4)

Dumps information about a variable

```
void var_dump (mixed expression)
```

This function returns structured information about an expression that includes its type and value. Arrays are explored recursively with values indented to show structure.

Compare **var_dump()** to **print_r()**.

```
<pre>
<?php
    $a = array (1, 2, array ("a", "b", "c"));
    var_dump ($a);
?>
</pre>
```


LXXIV. WDDX functions

These functions are intended for work with WDDX (<http://www.wddx.org/>).

In order to use WDDX, you will need to install the expat library (which comes with apache 1.3.7 or higher) and recompile PHP with `-with-xml` and `-enable-wddx`.

Note that all the functions that serialize variables use the first element of an array to determine whether the array is to be serialized into an array or structure. If the first element has string key, then it is serialized into a structure, otherwise, into an array.

Example 1. Serializing a single value

```
<?php
print wddx_serialize_value("PHP to WDDX packet example", "PHP packet");
?>
```

This example will produce:

```
<wddxPacket version='1.0'><header comment='PHP packet' /><data>
<string>PHP to WDDX packet example</string></data></wddxPacket>
```

Example 2. Using incremental packets

```
<?php
$pi = 3.1415926;
$packet_id = wddx_packet_start("PHP");
wddx_add_vars($packet_id, "pi");

/* Suppose $cities came from database */
$cities = array("Austin", "Novato", "Seattle");
wddx_add_vars($packet_id, "cities");

$packet = wddx_packet_end($packet_id);
print $packet;
?>
```

This example will produce:

```
<wddxPacket version='1.0'><header comment='PHP' /><data><struct>
<var name='pi'><number>3.1415926</number></var><var name='cities'>
<array length='3'><string>Austin</string><string>Novato</string>
<string>Seattle</string></array></var></struct></data></wddxPacket>
```


wddx_serialize_value (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Serialize a single value into a WDDX packet

```
string wddx_serialize_value (mixed var [, string comment])
```

wddx_serialize_value() is used to create a WDDX packet from a single given value. It takes the value contained in *var*, and an optional *comment* string that appears in the packet header, and returns the WDDX packet.

wddx_serialize_vars (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Serialize variables into a WDDX packet

```
string wddx_serialize_vars (mixed var_name [, mixed ...])
```

wddx_serialize_vars() is used to create a WDDX packet with a structure that contains the serialized representation of the passed variables.

wddx_serialize_vars() takes a variable number of arguments, each of which can be either a string naming a variable or an array containing strings naming the variables or another array, etc.

Example 1. wddx_serialize_vars example

```
<?php
$a = 1;
$b = 5.5;
$c = array("blue", "orange", "violet");
$d = "colors";

$c1vars = array("c", "d");
print wddx_serialize_vars("a", "b", $c1vars);
?>
```

The above example will produce:

```
<wddxPacket version='1.0'><header/><data><struct><var name='a'><number>1</number></var>
<var name='b'><number>5.5</number></var><var name='c'><array length='3'>
<string>blue</string><string>orange</string><string>violet</string></array></var>
<var name='d'><string>colors</string></var></struct></data></wddxPacket>
```

wddx_packet_start (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Starts a new WDDX packet with structure inside it

```
int wddx_packet_start ([string comment])
```

Use **wddx_packet_start()** to start a new WDDX packet for incremental addition of variables. It takes an optional *comment* string and returns a packet ID for use in later functions. It automatically creates a structure definition inside the packet to contain the variables.

wddx_packet_end (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Ends a WDDX packet with the specified ID

```
string wddx_packet_end (int packet_id)
```

wddx_packet_end() ends the WDDX packet specified by the *packet_id* and returns the string with the packet.

wddx_add_vars (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Add variables to a WDDX packet with the specified ID

```
wddx_add_vars (int packet_id, mixed name_var [, mixed ...])
```

wddx_add_vars() is used to serialize passed variables and add the result to the packet specified by the *packet_id*. The variables to be serialized are specified in exactly the same way as **wddx_serialize_vars()**.

wddx_deserialize (PHP 3>= 3.0.7, PHP 4 >= 4.0b2)

Deserializes a WDDX packet

```
mixed wddx_deserialize (string packet)
```

wddx_deserialize() takes a *packet* string and deserializes it. It returns the result which can be string, number, or array. Note that structures are deserialized into associative arrays.

LXXV. XML parser functions

Introduction

About XML

XML (eXtensible Markup Language) is a data format for structured document interchange on the Web. It is a standard defined by The World Wide Web consortium (W3C). Information about XML and related technologies can be found at <http://www.w3.org/XML/>.

Installation

This extension uses expat, which can be found at <http://www.jclark.com/xml/>. The Makefile that comes with expat does not build a library by default, you can use this make rule for that:

```
libexpat.a: $(OBJS)
    ar -rc $@ $(OBJS)
    ranlib $@
```

A source RPM package of expat can be found at <http://www.guardian.no/~ssb/phpxml.html>.

Note that if you are using Apache-1.3.7 or later, you already have the required expat library. Simply configure PHP using `-with-xml` (without any additional path) and it will automatically use the expat library built into Apache.

On UNIX, run **configure** with the `-with-xml` option. The expat library should be installed somewhere your compiler can find it. If you compile PHP as a module for Apache 1.3.9 or later, PHP will automatically use the bundled expat library from Apache. You may need to set `CPPFLAGS` and `LDLFLAGS` in your environment before running `configure` if you have installed expat somewhere exotic.

Build PHP. *Tada!* That should be it.

About This Extension

This PHP extension implements support for James Clark's expat in PHP. This toolkit lets you parse, but not validate, XML documents. It supports three source [character encodings](#) also provided by PHP: `US-ASCII`, `ISO-8859-1` and `UTF-8`. `UTF-16` is not supported.

This extension lets you [create XML parsers](#) and then define *handlers* for different XML events. Each XML parser also has a few [parameters](#) you can adjust.

The XML event handlers defined are:

Table 1. Supported XML handlers

PHP function to set handler	Event description
<code>xml_set_element_handler()</code>	Element events are issued whenever the XML parser encounters start or end tags. There are separate handlers for start tags and end tags.
<code>xml_set_character_data_handler()</code>	Character data is roughly all the non-markup contents of XML documents, including whitespace between tags. Note that the XML parser does not add or remove any whitespace, it is up to the application (you) to decide whether whitespace is significant.

PHP function to set handler	Event description
<code>xml_set_processing_instruction_handler()</code>	PHP programmers should be familiar with processing instructions (PIs) already. <code><?php ?></code> is a processing instruction, where <i>php</i> is called the "PI target". The handling of these are application-specific, except that all PI targets starting with "XML" are reserved.
<code>xml_set_default_handler()</code>	What goes not to another handler goes to the default handler. You will get things like the XML and document type declarations in the default handler.
<code>xml_set_unparsed_entity_decl_handler()</code>	This handler will be called for declaration of an unparsed (NDATA) entity.
<code>xml_set_notation_decl_handler()</code>	This handler is called for declaration of a notation.
<code>xml_set_external_entity_ref_handler()</code>	This handler is called when the XML parser finds a reference to an external parsed general entity. This can be a reference to a file or URL, for example. See the external entity example for a demonstration.

Case Folding

The element handler functions may get their element names *case-folded*. Case-folding is defined by the XML standard as "a process applied to a sequence of characters, in which those identified as non-uppercase are replaced by their uppercase equivalents". In other words, when it comes to XML, case-folding simply means uppercasing.

By default, all the element names that are passed to the handler functions are case-folded. This behaviour can be queried and controlled per XML parser with the `xml_parser_get_option()` and `xml_parser_set_option()` functions, respectively.

Error Codes

The following constants are defined for XML error codes (as returned by `xml_parse()`):

```
XML_ERROR_NONE
XML_ERROR_NO_MEMORY
XML_ERROR_SYNTAX
XML_ERROR_NO_ELEMENTS
XML_ERROR_INVALID_TOKEN
XML_ERROR_UNCLOSED_TOKEN
XML_ERROR_PARTIAL_CHAR
XML_ERROR_TAG_MISMATCH
XML_ERROR_DUPLICATE_ATTRIBUTE
XML_ERROR_JUNK_AFTER_DOC_ELEMENT
XML_ERROR_PARAM_ENTITY_REF
XML_ERROR_UNDEFINED_ENTITY
XML_ERROR_RECURSIVE_ENTITY_REF
XML_ERROR_ASYNC_ENTITY
XML_ERROR_BAD_CHAR_REF
XML_ERROR_BINARY_ENTITY_REF
```

```
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
XML_ERROR_MISPLACED_XML_PI
XML_ERROR_UNKNOWN_ENCODING
XML_ERROR_INCORRECT_ENCODING
XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING
```

Character Encoding

PHP's XML extension supports the Unicode (<http://www.unicode.org/>) character set through different *character encodings*. There are two types of character encodings, *source encoding* and *target encoding*. PHP's internal representation of the document is always encoded with UTF-8.

Source encoding is done when an XML document is [parsed](#). Upon [creating an XML parser](#), a source encoding can be specified (this encoding can not be changed later in the XML parser's lifetime). The supported source encodings are ISO-8859-1, US-ASCII and UTF-8. The former two are single-byte encodings, which means that each character is represented by a single byte. UTF-8 can encode characters composed by a variable number of bits (up to 21) in one to four bytes. The default source encoding used by PHP is ISO-8859-1.

Target encoding is done when PHP passes data to XML handler functions. When an XML parser is created, the target encoding is set to the same as the source encoding, but this may be changed at any point. The target encoding will affect character data as well as tag names and processing instruction targets.

If the XML parser encounters characters outside the range that its source encoding is capable of representing, it will return an error.

If PHP encounters characters in the parsed XML document that can not be represented in the chosen target encoding, the problem characters will be "demoted". Currently, this means that such characters are replaced by a question mark.

Some Examples

Here are some example PHP scripts parsing XML documents.

XML Element Structure Example

This first example displays the stucture of the start elements in a document with indentation.

Example 1. Show XML Element Structure

```
$file = "data.xml";
$depth = array();

function startElement($parser, $name, $attrs) {
    global $depth;
    for ($i = 0; $i < $depth[$parser]; $i++) {
        print "    ";
    }
    print "$name\n";
    $depth[$parser]++;
}

function endElement($parser, $name) {
    global $depth;
    $depth[$parser]--;
}
```

```

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
xml_parser_free($xml_parser);

```

XML Tag Mapping Example

Example 2. Map XML to HTML

This example maps tags in an XML document directly to HTML tags. Elements not found in the "map array" are ignored. Of course, this example will only work with a specific XML document type.

```

$file = "data.xml";
$map_array = array(
    "BOLD"      => "B",
    "EMPHASIS"  => "I",
    "LITERAL"   => "TT"
);

function startElement($parser, $name, $attrs) {
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "<$htmltag>";
    }
}

function endElement($parser, $name) {
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "</>$htmltag";
    }
}

function characterData($parser, $data) {
    print $data;
}

$xml_parser = xml_parser_create();
// use case-folding so we are sure to find the tag in $map_array
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {

```

```

        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
xml_parser_free($xml_parser);

```

XML External Entity Example

This example highlights XML code. It illustrates how to use an external entity reference handler to include and parse other documents, as well as how PIs can be processed, and a way of determining "trust" for PIs containing code.

XML documents that can be used for this example are found below the example (`xmltest.xml` and `xmltest2.xml`).

Example 3. External Entity Example

```

$file = "xmltest.xml";

function trustedFile($file) {
    // only trust local files owned by ourselves
    if (!eregi("^([a-z]+://", $file)
        && fileowner($file) == getmyuid()) {
        return true;
    }
    return false;
}

function startElement($parser, $name, $attribs) {
    print "<".$name.">";
    if (sizeof($attribs)) {
        while (list($k, $v) = each($attribs)) {
            print " ".$k."=".$v;
        }
    }
    print ">";
}

function endElement($parser, $name) {
    print "</".$name.">";
}

function characterData($parser, $data) {
    print "<b>$data</b>";
}

function PIHandler($parser, $target, $data) {
    switch (strtolower($target)) {
        case "php":
            global $parser_file;
            // If the parsed document is "trusted", we say it is safe
            // to execute PHP code inside it. If not, display the code
            // instead.
            if (trustedFile($parser_file[$parser])) {
                eval($data);
            } else {

```

```

        printf("Untrusted PHP code: <i>%s</i>",
               htmlspecialchars($data));
    }
    break;
}
}

function defaultHandler($parser, $data) {
    if (substr($data, 0, 1) == "&" && substr($data, -1, 1) == ";") {
        printf('<font color="#aa00aa">%s</font>',
               htmlspecialchars($data));
    } else {
        printf('<font size="-1">%s</font>',
               htmlspecialchars($data));
    }
}

function externalEntityRefHandler($parser, $openEntityNames, $base, $systemId,
                                  $publicId) {
    if ($systemId) {
        if (!list($parser, $fp) = new_xml_parser($systemId)) {
            printf("Could not open entity %s at %s\n", $openEntityNames,
                   $systemId);
            return false;
        }
        while ($data = fread($fp, 4096)) {
            if (!xml_parse($parser, $data, feof($fp))) {
                printf("XML error: %s at line %d while parsing entity %s\n",
                       xml_error_string(xml_get_error_code($parser)),
                       xml_get_current_line_number($parser), $openEntityNames);
                xml_parser_free($parser);
                return false;
            }
        }
        xml_parser_free($parser);
        return true;
    }
    return false;
}

function new_xml_parser($file) {
    global $parser_file;

    $xml_parser = xml_parser_create();
    xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, 1);
    xml_set_element_handler($xml_parser, "startElement", "endElement");
    xml_set_character_data_handler($xml_parser, "characterData");
    xml_set_processing_instruction_handler($xml_parser, "PIHandler");
    xml_set_default_handler($xml_parser, "defaultHandler");
    xml_set_external_entity_ref_handler($xml_parser, "externalEntityRefHandler");

    if (!($fp = @fopen($file, "r"))) {
        return false;
    }
    if (!is_array($parser_file)) {
        settype($parser_file, "array");
    }
    $parser_file[$xml_parser] = $file;
    return array($xml_parser, $fp);
}

```

```

if (!(list($xml_parser, $fp) = new_xml_parser($file))) {
    die("could not open XML input");
}

print "<pre>";
while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d\n",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
print "</pre>";
print "parse complete\n";
xml_parser_free($xml_parser);

?>

```

Example 4. xmltest.xml

```

<?xml version='1.0'?>
<!DOCTYPE chapter SYSTEM "/just/a/test.dtd" [
<!ENTITY plainEntity "FOO entity">
<!ENTITY systemEntity SYSTEM "xmltest2.xml">
]!
<chapter>
<TITLE>Title &plainEntity;</TITLE>
<para>
<informaltable>
<tgroup cols="3">
<tbody>
<row><entry>a1</entry><entry morerows="1">b1</entry><entry>c1</entry></row>
<row><entry>a2</entry><entry>c2</entry></row>
<row><entry>a3</entry><entry>b3</entry><entry>c3</entry></row>
</tbody>
</tgroup>
</informaltable>
</para>
&systemEntity;
<sect1 id="about">
<title>About this Document</title>
<para>
<!-- this is a comment -->
<?php print 'Hi! This is PHP version '.phpversion(); ?>
</para>
</sect1>
</chapter>

```

This file is included from `xmltest.xml`:

Example 5. xmltest2.xml

```

<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY testEnt "test entity">
]!

```

```
<foo>
  <element attrib="value" />
  &testEnt;
  <?php print "This is some more PHP code being executed."; ?>
</foo>
```

xml_parser_create (PHP 3>= 3.0.6, PHP 4)

create an XML parser

```
int xml_parser_create ([string encoding])
```

encoding (optional)

Which character encoding the parser should use. The following character encodings are supported:

ISO-8859-1 (default)

US-ASCII

UTF-8

This function creates an XML parser and returns a handle for use by other XML functions. Returns `false` on failure.

xml_set_object (PHP 4 >= 4.0b4)

Use XML Parser within an object

```
void xml_set_object (int parser, object &object)
```

This function allows to use *parser* inside *object*. All callback functions could be set with `xml_set_element_handler()` etc and assumed to be methods of *object*.

```
<?php
class xml {
var $parser;

function xml() {
    $this->parser = xml_parser_create();
    xml_set_object($this->parser,&$this);
    xml_set_element_handler($this->parser,"tag_open","tag_close");
    xml_set_character_data_handler($this->parser,"cdata");
}

function parse($data) {
    xml_parse($this->parser,$data);
}

function tag_open($parser,$tag,$attributes) {
    var_dump($parser,$tag,$attributes);
}

function cdata($parser,$cdata) {
    var_dump($parser,$cdata);
}

function tag_close($parser,$tag) {
    var_dump($parser,$tag);
}

} // end of class xml

$xml_parser = new xml();
$xml_parser->parse("<A ID=\\"hallo\\">PHP</A>");
```

?>

xml_set_element_handler (PHP 3>= 3.0.6, PHP 4)

set up start and end element handlers

```
int xml_set_element_handler (int parser, string startElementHandler, string  

endElementHandler)
```

Sets the element handler functions for the XML parser *parser*. *startElementHandler* and *endElementHandler* are strings containing the names of functions that must exist when **xml_parse()** is called for *parser*.

The function named by *startElementHandler* must accept three parameters:

```
startElementHandler (int parser, string name, array attribs)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

name

The second parameter, *name*, contains the name of the element for which this handler is called. If [case-folding](#) is in effect for this parser, the element name will be in uppercase letters.

attribs

The third parameter, *attribs*, contains an associative array with the element's attributes (if any). The keys of this array are the attribute names, the values are the attribute values. Attribute names are [case-folded](#) on the same criteria as element names. Attribute values are *not* case-folded.

The original order of the attributes can be retrieved by walking through *attribs* the normal way, using **each()**. The first key in the array was the first attribute, and so on.

The function named by *endElementHandler* must accept two parameters:

```
endElementHandler (int parser, string name)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

name

The second parameter, *name*, contains the name of the element for which this handler is called. If [case-folding](#) is in effect for this parser, the element name will be in uppercase letters.

If a handler function is set to an empty string, or **false**, the handler in question is disabled.

True is returned if the handlers are set up, **false** if *parser* is not a parser.

There is currently no support for object/method handlers. See **xml_set_object()** for using the XML parser within an object.

xml_set_character_data_handler (PHP 3>= 3.0.6, PHP 4)

set up character data handler

```
int xml_set_character_data_handler ( int parser, string handler )
```

Sets the character data handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **xml_parse()** is called for *parser*.

The function named by *handler* must accept two parameters:

```
handler ( int parser, string data )
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

data

The second parameter, *data*, contains the character data as a string.

If a handler function is set to an empty string, or **false**, the handler in question is disabled.

True is returned if the handler is set up, **false** if *parser* is not a parser.

There is currently no support for object/method handlers. See **xml_set_object()** for using the XML parser within an object.

xml_set_processing_instruction_handler (PHP 3>= 3.0.6, PHP 4)

Set up processing instruction (PI) handler

```
int xml_set_processing_instruction_handler ( int parser, string handler )
```

Sets the processing instruction (PI) handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **xml_parse()** is called for *parser*.

A processing instruction has the following format:

```
<?
  target
  data?>
```

You can put PHP code into such a tag, but be aware of one limitation: in an XML PI, the PI end tag (**?>**) can not be quoted, so this character sequence should not appear in the PHP code you embed with PIs in XML documents. If it does, the rest of the PHP code, as well as the "real" PI end tag, will be treated as character data.

The function named by *handler* must accept three parameters:

```
handler ( int parser, string target, string data )
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

target

The second parameter, *target*, contains the PI target.

data

The third parameter, *data*, contains the PI data.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

`True` is returned if the handler is set up, `false` if *parser* is not a parser.

There is currently no support for object/method handlers. See **`xml_set_object()`** for using the XML parser within an object.

`xml_set_default_handler` (PHP 3>= 3.0.6, PHP 4)

set up default handler

```
int xml_set_default_handler (int parser, string handler)
```

Sets the default handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **`xml_parse()`** is called for *parser*.

The function named by *handler* must accept two parameters:

```
handler (int parser, string data)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

data

The second parameter, *data*, contains the character data. This may be the XML declaration, document type declaration, entities or other data for which no other handler exists.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

`True` is returned if the handler is set up, `false` if *parser* is not a parser.

There is currently no support for object/method handlers. See **`xml_set_object()`** for using the XML parser within an object.

`xml_set_unparsed_entity_decl_handler` (PHP 3>= 3.0.6, PHP 4)

Set up unparsed entity declaration handler

```
int xml_set_unparsed_entity_decl_handler (int parser, string handler)
```

Sets the unparsed entity declaration handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **`xml_parse()`** is called for *parser*.

This handler will be called if the XML parser encounters an external entity declaration with an NDATA declaration, like the following:

```
<!ENTITY name {publicId | systemId}
```

```
NDATA notationName>
```

See section 4.2.2 of the XML 1.0 spec (<http://www.w3.org/TR/1998/REC-xml-19980210#sec-external-ent>) for the definition of notation declared external entities.

The function named by *handler* must accept six parameters:

```
handler (int parser, string entityName, string base, string systemId, string publicId,
        string notationName)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

entityName

The name of the entity that is about to be defined.

base

This is the base for resolving the system identifier (*systemId*) of the external entity. Currently this parameter will always be set to an empty string.

systemId

System identifier for the external entity.

publicId

Public identifier for the external entity.

notationName

Name of the notation of this entity (see **xml_set_notation_decl_handler()**).

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

`True` is returned if the handler is set up, `false` if *parser* is not a parser.

There is currently no support for object/method handlers. See **xml_set_object()** for using the XML parser within an object.

xml_set_notation_decl_handler (PHP 3>= 3.0.6, PHP 4)

set up notation declaration handler

```
int xml_set_notation_decl_handler (int parser, string handler)
```

Sets the notation declaration handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **xml_parse()** is called for *parser*.

A notation declaration is part of the document's DTD and has the following format:

```
<!NOTATION
  name {systemId | publicId}>
```

See section 4.7 of the XML 1.0 spec (<http://www.w3.org/TR/1998/REC-xml-19980210#Notations>) for the definition of notation declarations.

The function named by *handler* must accept five parameters:

```
handler (int parser, string notationName, string base, string systemId, string publicId)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

notationName

This is the notation's *name*, as per the notation format described above.

base

This is the base for resolving the system identifier (*systemId*) of the notation declaration. Currently this parameter will always be set to an empty string.

systemId

System identifier of the external notation declaration.

publicId

Public identifier of the external notation declaration.

If a handler function is set to an empty string, or *false*, the handler in question is disabled.

True is returned if the handler is set up, *false* if *parser* is not a parser.

There is currently no support for object/method handlers. See **xml_set_object()** for using the XML parser within an object.

xml_set_external_entity_ref_handler (PHP 3>= 3.0.6, PHP 4)

set up external entity reference handler

```
int xml_set_external_entity_ref_handler (int parser, string handler)
```

Sets the notation declaration handler function for the XML parser *parser*. *handler* is a string containing the name of a function that must exist when **xml_parse()** is called for *parser*.

The function named by *handler* must accept five parameters, and should return an integer value. If the value returned from the handler is *false* (which it will be if no value is returned), the XML parser will stop parsing and **xml_get_error_code()** will return **XML_ERROR_EXTERNAL_ENTITY_HANDLING**.

```
int handler (int parser, string openEntityNames, string base, string systemId, string publicId)
```

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

openEntityNames

The second parameter, *openEntityNames*, is a space-separated list of the names of the entities that are open for the parse of this entity (including the name of the referenced entity).

base

This is the base for resolving the system identifier (*systemId*) of the external entity. Currently this parameter will always be set to an empty string.

systemId

The fourth parameter, *systemId*, is the system identifier as specified in the entity declaration.

publicId

The fifth parameter, *publicId*, is the public identifier as specified in the entity declaration, or an empty string if none was specified; the whitespace in the public identifier will have been normalized as required by the XML spec.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

`True` is returned if the handler is set up, `false` if *parser* is not a parser.

There is currently no support for object/method handlers. See **xml_set_object()** for using the XML parser within an object.

xml_parse (PHP 3>= 3.0.6, PHP 4)

start parsing an XML document

```
int xml_parse (int parser, string data [, int isFinal])
```

parser

A reference to the XML parser to use.

data

Chunk of data to parse. A document may be parsed piece-wise by calling **xml_parse()** several times with new data, as long as the *isFinal* parameter is set and true when the last data is parsed.

***isFinal* (optional)**

If set and true, *data* is the last piece of data sent in this parse.

When the XML document is parsed, the handlers for the configured events are called as many times as necessary, after which this function returns true or false.

`True` is returned if the parse was successful, `false` if it was not successful, or if *parser* does not refer to a valid parser. For unsuccessful parses, error information can be retrieved with **xml_get_error_code()**, **xml_error_string()**, **xml_get_current_line_number()**, **xml_get_current_column_number()** and **xml_get_current_byte_index()**.

xml_get_error_code (PHP 3>= 3.0.6, PHP 4)

get XML parser error code

```
int xml_get_error_code (int parser)
```

parser

A reference to the XML parser to get error code from.

This function returns false if *parser* does not refer to a valid parser, or else it returns one of the error codes listed in the [error codes section](#).

xml_error_string (PHP 3>= 3.0.6, PHP 4)

get XML parser error string

```
string xml_error_string ( int code )
```

code

An error code from **xml_get_error_code()**.

Returns a string with a textual description of the error code *code*, or false if no description was found.

xml_get_current_line_number (PHP 3>= 3.0.6, PHP 4)

get current line number for an XML parser

```
int xml_get_current_line_number ( int parser )
```

parser

A reference to the XML parser to get line number from.

This function returns false if *parser* does not refer to a valid parser, or else it returns which line the parser is currently at in its data buffer.

xml_get_current_column_number (PHP 3>= 3.0.6, PHP 4)

Get current column number for an XML parser

```
int xml_get_current_column_number ( int parser )
```

parser

A reference to the XML parser to get column number from.

This function returns false if *parser* does not refer to a valid parser, or else it returns which column on the current line (as given by **xml_get_current_line_number()**) the parser is currently at.

xml_get_current_byte_index (PHP 3>= 3.0.6, PHP 4)

get current byte index for an XML parser

```
int xml_get_current_byte_index ( int parser )
```

parser

A reference to the XML parser to get byte index from.

This function returns false if *parser* does not refer to a valid parser, or else it returns which byte index the parser is currently at in its data buffer (starting at 0).

xml_parse_into_struct (PHP 3>= 3.0.8, PHP 4)

Parse XML data into an array structure

```
int xml_parse_into_struct ( int parser, string data, array &values, array &index )
```

This function parses an XML file into 2 parallel array structures, one (*index*) containing pointers to the location of the appropriate values in the *values* array. These last two parameters must be passed by reference.

Below is an example that illustrates the internal structure of the arrays being generated by the function. We use a simple note tag embeded inside a para tag, and then we parse this an print out the structures generated:

```
$simple = "<para><note>simple note</note></para>" ;
$p = xml_parser_create();
xml_parse_into_struct($p,$simple,$vals,$index);
xml_parser_free($p);
echo "Index array\n";
print_r($index);
echo "\nVals array\n";
print_r($vals);
```

When we run that code, the output will be:

```
Index array
Array
(
    [PARA] => Array
        (
            [0] => 0
            [1] => 2
        )

    [NOTE] => Array
        (
            [0] => 1
        )
)

Vals array
Array
```

```

(
  [0] => Array
  (
    [tag] => PARA
    [type] => open
    [level] => 1
  )

  [1] => Array
  (
    [tag] => NOTE
    [type] => complete
    [level] => 2
    [value] => simple note
  )

  [2] => Array
  (
    [tag] => PARA
    [type] => close
    [level] => 1
  )
)

```

Event-driven parsing (based on the expat library) can get complicated when you have an XML document that is complex. This function does not produce a DOM style object, but it generates structures amenable of being transversed in a tree fashion. Thus, we can create objects representing the data in the XML file easily. Let's consider the following XML file representing a small database of aminoacids information:

Example 1. moldb.xml - small database of molecular information

```

<?xml version="1.0"?>
<moldb>

  <molecule>
    <name>Alanine</name>
    <symbol>ala</symbol>
    <code>A</code>
    <type>hydrophobic</type>
  </molecule>

  <molecule>
    <name>Lysine</name>
    <symbol>lys</symbol>
    <code>K</code>
    <type>charged</type>
  </molecule>

</moldb>

```

And some code to parse the document and generate the appropriate objects:

Example 2. parsemoldb.php - parses moldb.xml into and array of molecular objects

```

<?php

class AminoAcid {

```

```

var $name; // aa name
var $symbol; // three letter symbol
var $code; // one letter code
var $type; // hydrophobic, charged or neutral

function AminoAcid ($aa) {
    foreach ($aa as $k=>$v)
        $this->$k = $aa[$k];
}

function readDatabase($filename) {
    // read the xml database of aminoacids
    $data = implode("",file($filename));
    $parser = xml_parser_create();
    xml_parser_set_option($parser,XML_OPTION_CASE_FOLDING,0);
    xml_parser_set_option($parser,XML_OPTION_SKIP_WHITE,1);
    xml_parse_into_struct($parser,$data,$values,$tags);
    xml_parser_free($parser);

    // loop through the structures
    foreach ($tags as $key=>$val) {
        if ($key == "molecule") {
            $molranges = $val;
            // each contiguous pair of array entries are the
            // lower and upper range for each molecule definition
            for ($i=0; $i < count($molranges); $i+=2) {
                $offset = $molranges[$i] + 1;
                $len = $molranges[$i + 1] - $offset;
                $tdb[] = parseMol(array_slice($values, $offset, $len));
            }
        } else {
            continue;
        }
    }
    return $tdb;
}

function parseMol($mvalues) {
    for ($i=0; $i < count($mvalues); $i++)
        $mol[$mvalues[$i]["tag"]] = $mvalues[$i]["value"];
    return new AminoAcid($mol);
}

$db = readDatabase("moldb.xml");
echo "** Database of AminoAcid objects:\n";
print_r($db);

?>

```

After executing `parsemoldb.php`, the variable `$db` contains an array of `AminoAcid` objects, and the output of the script confirms that:

```

** Database of AminoAcid objects:
Array
(
    [0] => aminoacid Object
        (
            [name] => Alanine
            [symbol] => ala
            [code] => A

```

```

        [type] => hydrophobic
    )

[1] => aminoacid Object
(
    [name] => Lysine
    [symbol] => lys
    [code] => K
    [type] => charged
)
)

```

xml_parser_free (PHP 3>= 3.0.6, PHP 4)

Free an XML parser

```
string xml_parser_free ( int parser )
```

parser

A reference to the XML parser to free.

This function returns false if *parser* does not refer to a valid parser, or else it frees the parser and returns true.

xml_parser_set_option (PHP 3>= 3.0.6, PHP 4)

set options in an XML parser

```
int xml_parser_set_option ( int parser, int option, mixed value )
```

parser

A reference to the XML parser to set an option in.

option

Which option to set. See below.

value

The option's new value.

This function returns false if *parser* does not refer to a valid parser, or if the option could not be set. Else the option is set and true is returned.

The following options are available:

Table 1. XML parser options

Option constant	Data type	Description
XML_OPTION_CASE_FOLDING	integer	Controls whether case-folding is enabled for this XML parser. Enabled by default.
XML_OPTION_TARGET_ENCODING	string	Sets which target encoding to use in this XML parser. By default, it is set to the same as the source encoding used by xml_parser_create() . Supported target encodings are ISO-8859-1 , US-ASCII and UTF-8 .

xml_parser_get_option (PHP 3>= 3.0.6, PHP 4)

get options from an XML parser

```
mixed xml_parser_get_option (int parser, int option)
```

parser

A reference to the XML parser to get an option from.

option

Which option to fetch. See [xml_parser_set_option\(\)](#) for a list of options.

This function returns false if *parser* does not refer to a valid parser, or if the option could not be set. Else the option's value is returned.

See [xml_parser_set_option\(\)](#) for the list of options.

utf8_decode (PHP 3>= 3.0.6, PHP 4)

Converts a string with ISO-8859-1 characters encoded with UTF-8 to single-byte ISO-8859-1.

```
string utf8_decode (string data)
```

This function decodes *data*, assumed to be UTF-8 encoded, to ISO-8859-1.

See [utf8_encode\(\)](#) for an explanation of UTF-8 encoding.

utf8_encode (PHP 3>= 3.0.6, PHP 4)

encodes an ISO-8859-1 string to UTF-8

```
string utf8_encode (string data)
```

This function encodes the string *data* to UTF-8, and returns the encoded version. UTF-8 is a standard mechanism used by Unicode for encoding wide character values into a byte stream. UTF-8 is transparent to plain ASCII characters, is

self-synchronized (meaning it is possible for a program to figure out where in the bytestream characters start) and can be used with normal string comparison functions for sorting and such. PHP encodes UTF-8 characters in up to four bytes, like this:

Table 1. UTF-8 encoding

bytes	bits	representation
1	7	0bbbbbb
2	11	110bbbb 10bbbb
3	16	1110bbb 10bbbb 10bbbb
4	21	11110bbb 10bbbb 10bbbb 10bbbb

Each *b* represents a bit that can be used to store character data.

LXXVI. XSLT functions

Introduction

About XSLT and Sablotron

XSLT (Extensible Stylesheet Language (XSL) Transformations) is a language for transforming XML documents into other XML documents. It is a standard defined by The World Wide Web consortium (W3C). Information about XSLT and related technologies can be found at <http://www.w3.org/TR/xslt>.

Installation

This extension uses Sabloton and expat, which can both be found at <http://www.gingerall.com/>. Binaries are provided as well as source.

On UNIX, run **configure** with the `-with-sablot`. The Sablotron library should be installed somewhere your compiler can find it.

About This Extension

This PHP extension implements support Sablotron from Ginger Alliance in PHP. This toolkit lets you transform XML documents into other documents, including new XML documents, but also into HTML or other target formats. It basically provides a standardized and portable template mechanism, separating content and design of a website.

xslt_closelog (PHP 4 >= 4.0.3)

Clear the logfile for a given instance of Sablotron

```
bool xslt_closelog (resource xh)
```

xh

A reference to the XSLT parser.

This function returns false if *parser* does not refer to a valid parser, or if the closing of the logfile fails. Otherwise it returns true.

xslt_create (PHP 4 >= 4.0.3)

Create a new XSL processor.

```
resource xslt_create(void);
```

This function returns a handle for a new XSL processor. This handle is needed in all subsequent calls to XSL functions.

xslt_errno (PHP 4 >= 4.0.3)

Return the current error number

```
int xslt_errno ([int xh])
```

Return the current error number of the given XSL processor. If no handle is given, the last error number that occurred anywhere is returned.

xslt_error (PHP 4 >= 4.0.3)

Return the current error string

```
mixed xslt_error ([int xh])
```

Return the current error string of the given XSL processor. If no handle is given, the last error string that occurred anywhere is returned.

xslt_fetch_result (PHP 4 >= 4.0.3)

Fetch a (named) result buffer

```
string xslt_fetch_result ([int xh string result_name])
```

Fetch a result buffer from the XSLT processor identified by the given handle. If no result name is given, the buffer named "/_result" is fetched.

xslt_free (PHP 4 >= 4.0.3)

Free XSLT processor

```
void xslt_free (resource xh)
```

Free the XSLT processor identified by the given handle.

xslt_openlog (PHP 4 >= 4.0.3)

Set a logfile for XSLT processor messages

```
bool xslt_openlog ([resource xh string logfile int loglevel])
```

Set a logfile for the XSLT processor to place all of its error messages.

xslt_output_begintransform (PHP 4 >= 4.0.3)

Begin an XSLT transformation on the output

```
void xslt_output_begintransform (string xslt_filename)
```

This function will begin the output transformation on your data. From the point you call **xslt_output_begintransform()** till the point you call **xslt_output_endtransform()** all output will be transformed through the xslt stylesheet given by the first argument.

Note: This function does not currently support nested output transformations.

Example 1. Transforming output through an XSLT stylesheet, using the DOM-XML functions for xml generation

```
<?php

$xsl_file = "article.xsl";
xslt_output_begintransform($xsl_file);

$doc = new_xmldoc('1.0');
$article = $doc->new_root('article');

$article->new_child('title', 'The History of South Tyrol');
$article->new_child('author', 'Sterling Hughes');
$article->new_child('body', 'Back after WWI, Italy gained South Tyrol from
                           Austria. Since that point nothing interesting has
                           happened');
```

```
echo $doc->dumpmem();
xslt_output_endtransform();
```

xslt_output_endtransform (PHP 4 >= 4.0.3)

End an output transformation started with xslt_output_begintransform

```
void xslt_output_endtransform (void);
```

The **xslt_output_endtransform()** ends the output transformation started by the **xslt_output_begintransform()** function. You must call this function in order to see the results of the output transformation.

xslt_process (PHP 4 >= 4.0.3)

Transform XML data through a string containing XSL data

```
bool xslt_process (string xsl_data string xml_data string result)
```

The **xslt_process()** takes a string containing the XSLT stylesheet as its first argument, it takes a second string containing the XML data you want to transform and then a third string containing the results of the transformation. **xslt_process()** will return true on success and false on failure, to get the error number and error string if an error occurs use the **xslt_errno()** and **xslt_error()** functions.

Example 1. Using the xslt_process() to transform three strings

```
<?php

$xslData = '
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="article">
    <table border="1" cellpadding="2" cellspacing="1">
        <tr>
            <td width="20%">

                </td>
            <td width="80%">
                <h2><xsl:value-of select="title"/></h2>
                <h3><xsl:value-of select="author"/></h3>
                <br/>

                <xsl:value-of select="body" />
            </td>
        </tr>
    </table>
</xsl:template>

</xsl:stylesheet>';
```

```
$xmlData = '
<?xml version="1.0"?>
<article>
    <title>Learning German</title>
    <author>Sterling Hughes</author>
    <body>
        Essential phrases:
        <br/>
        <br/>
        Komme sie mir sagen, woe die toilette es?<br/>
        Eine grosse beer bitte!<br/>
        Noch einem bitte.<br/>
    </body>
</article>';

if (xslt_process($xslData, $xmlData, $result))
{
    echo "Here is the brilliant in-depth article on learning";
    echo " German: ";
    echo "<br>\n<br>";
    echo $result;
}
else
{
    echo "There was an error that occurred in the XSL transformation...\n";
    echo "\tError number: " . xslt_errno() . "\n";
    echo "\tError string: " . xslt_error() . "\n";
    exit;
}
?>
```

xslt_run (PHP 4 >= 4.0.3)

Apply a XSLT stylesheet to a file.

```
bool xslt_run ([resource xh string xslt_file string xml_data_file string result array xslt_params array xslt_args]])
```

Process the *xml_data_file* by applying the *xslt_file* stylesheet to it. The stylesheet has access to *xslt_params* and the processor is started with *xslt_args*. The result of the XSLT transformation is placed in the named buffer (default is "/*result*").

xslt_set_sax_handler (PHP 4 >= 4.0.3)

Set SAX handlers for a XSLT processor

```
bool xslt_set_sax_handler (resource xh array handlers)
```

Set SAX handlers on the resource handle given by *xh*.

xslt_transform (PHP 4 >= 4.0.3)

Perform an XSLT transformation

```
bool xslt_transform (string xsl string xml string result string params string args string  
resultBuffer)
```

The **xslt_transform()** provides an interface to sablotron's more advanced features without requiring you to use the resource API.

LXXVII. YAZ functions

Introduction

This extension offers a PHP interface to the YAZ toolkit that implements the Z39.50 protocol for information retrieval. With this extension you can easily implement a Z39.50 origin (client) that searches Z39.50 targets (servers) in parallel.

YAZ is available at <http://www.indexdata.dk/yaz/>. You can find news information, example scripts, etc. for this extension at <http://www.indexdata.dk/phpyaz/>.

The module hides most of the complexity of Z39.50 so it should be fairly easy to use. It supports persistent stateless connections very similar to those offered by the various SQL APIs that are available for PHP. This means that sessions are stateless but shared amongst users, thus saving the connect and initialize phase steps in most cases.

Installation

Compile YAZ and install it. Build PHP with your favourite modules and add option `--with-yaz`. Your task is roughly the following:

```
gunzip -c yaz-1.6.tar.gz|tar xf -
gunzip -c php-4.0.X.tar.gz|tar xf -
cd yaz-1.6
./configure --prefix=/usr
make
make install
cd ../php-4.0.X
./configure --with-yaz=/usr/bin
make
make install
```

Example

PHP/YAZ keeps track of connections with targets (Z-Associations). A positive integer represents the ID of a particular association.

The script below demonstrates the parallel searching feature of the API. When invoked with no arguments it prints a query form; else (arguments are supplied) it searches the targets as given in array host.

Example 1. YAZ()

```
$num_hosts = count ($host);
if (empty($term) || count($host) == 0) {
    echo '<form method="get">
<input type="checkbox"
name="host[]"' value="bagel.indexdata.dk/gils">
    GILS test
<input type="checkbox"
name="host[]"' value="localhost:9999/Default">
    local test
<input type="checkbox" checked="1"
name="host[]"' value="z3950.bell-labs.com/books">
    BELL Labs Library
<br>
RPN Query:'
```

```
<input type="text" size="30" name="term">
<input type="submit" name="action" value="Search">
';
} else {
    echo 'You searched for ' . htmlspecialchars($term) . '<br>';
    for ($i = 0; $i < $num_hosts; $i++) {
        $id[] = yaz_connect($host[$i]);
        yaz_syntax($id[$i], "sutrs");
        yaz_search($id[$i], "rpn", $term);
    }
    yaz_wait();
    for ($i = 0; $i < $num_hosts; $i++) {
        echo '<hr>' . $host[$i] . ":" ;
        $error = yaz_error($id[$i]);
        if (!empty($error)) {
            echo "Error: $error";
        } else {
            $hits = yaz_hits($id[$i]);
            echo "Result Count $hits";
        }
        echo '<dl>';
        for ($p = 1; $p <= 10; $p++) {
            $rec = yaz_record($id[$i], $p, "string");
            if (empty($rec)) continue;
            echo "<dt><b>$p</b></dt><dd>";
            echo ereg_replace("\n", "<br>\n", $rec);
            echo "</dd>";
        }
        echo '</dl>';
    }
}
```

yaz_addinfo (PHP 4 >= 4.0.1)

Returns additional error information

```
int yaz_addinfo (int id)
```

Returns additional error message for target (last request). An empty string is returned if last operation was a success or if no additional information was provided by the target.

yaz_close (PHP 4 >= 4.0.1)

Closes a YAZ connection

```
int yaz_close (int id)
```

Closes a connection to a target. The application can no longer refer to the target with the given ID.

yaz_connect (PHP 4 >= 4.0.1)

Returns a positive association ID on success; zero on failure

```
int yaz_connect (string zurl)
```

Yaz_connect() prepares for a connection to a Z39.50 target. The zurl argument takes the form host[:port][/database]. If port is omitted 210 is used. If database is omitted Default is used. This function is non-blocking and doesn't attempt to establish a socket - it merely prepares a connect to be performed later when **yaz_wait()** is called.

yaz_errno (PHP 4 >= 4.0.1)

Returns error number

```
int yaz_errno (int id)
```

Returns error for target (last request). A positive value is returned if the target returned a diagnostic code; a value of zero is returned if no errors occurred (success); negative value is returned for other errors targets didn't indicate the error in question.

Yaz_errno() should be called after network activity for each target - (after **yaz_wait()** returns) to determine the success or failure of the last operation (e.g. search).

yaz_error (PHP 4 >= 4.0.1)

Returns error description

```
int yaz_error (int id)
```

Returns error message for target (last request). An empty string is returned if last operation was a success.

Yaz_error() returns a english message corresponding to the last error number as returned by **yaz_errno()**.

yaz_hits (PHP 4 >= 4.0.1)

Returns number of hits for last search

```
int yaz_hits (int id)
```

Yaz_hits() returns number of hits for last search.

yaz_element (PHP 4 >= 4.0.1)

Specifies Element-Set Name for retrieval

```
int yaz_range (int id, string elementset)
```

This function is used in conjunction with **yaz_search()** to specify the element set name for records to be retrieved. Most servers support F (full) and B (brief).

Returns true on success; false on error.

yaz_range (PHP 4 >= 4.0.1)

Specifies the maximum number of records to retrieve

```
int yaz_range (int id, int start, int number)
```

This function is used in conjunction with **yaz_search()** to specify the maximum number of records to retrieve (number) and the first record position (start). If this function is not invoked (only **yaz_search()**) start is set to 1 and number is set to 10.

Returns true on success; false on error.

yaz_record (PHP 4 >= 4.0.1)

Returns a record

```
int yaz_record (int id, int pos, string type)
```

Returns record at position or empty string if no record exists at given position.

The **yaz_record()** function inspects a record in the current result set at the position specified. If no database record exists at the given position an empty string is returned. The argument, type, specifies the form of the returned record. If type is "string" the record is returned in a string representation suitable for printing (for XML and SUTRS). If type is "array" the record is returned as an array representation (for structured records).

yaz_search (PHP 4 >= 4.0.1)

Prepares for a search

```
int yaz_search (int id, string type, string query)
```

yaz_search() prepares for a search on the target with given id. The type represents the query type - only "rpn" is supported now in which case the third argument specifies a Type-1 query (RPN). Like **yaz_connect()** this function is non-blocking and only prepares for a search to be executed later when **yaz_wait()** is called.

The RPN query is a textual representation of the Type-1 query as defined by the Z39.50 standard. However, in the text representation as used by YAZ a prefix notation is used, that is the operator precedes the operands. The query string is a sequence of tokens where white space is ignored is ignored unless surrounded by double quotes. Tokens beginning with an at-character (@) are considered operators, otherwise they are treated as search terms.

Table 1. RPN Operators

Syntax	Description
@and query1 query2	intersection of query1 and query2
@or query1 query2	union of query1 and query2
@not query1 query2	query1 and not query2
@set name	result set reference
@attrset set query	specifies attribute-set for query. This construction is only allowed once - in the beginning of the whole query
@attr set type=value query	applies attribute to query. The type and value are integers specifying the attribute-type and attribute-value respectively. The set, if given, specifies the attribute-set.

The following illustrates valid query constructions:

computer

Matches documents where "computer" occur. No attributes are specified.

"donald knuth"

Matches documents where "donald knuth" occur.

@attr 1=4 art

Attribute type is 1 (Bib-1 use), attribute value is 4 Title), so this should match documents where "art" occur in the title.

@attrset gils @and @attr 1=4 art @attr 1=1003 "donald knuth"

The query as a whole uses the GILS attributeset. The query matches documents where "art" occur in the title and in which "donald knuth" occur in the author.

yaz_syntax (PHP 4 >= 4.0.1)

Specifies the object identifier (OID) for the preferred record syntax for retrieval.

```
int yaz_syntax (int id, string syntax)
```

The syntax may be specified in a raw dot-notation (like 1.2.840.10003.5.10) or as one of the known record syntaxes (sutrs, usmarc, grs1, xml, etc.). This function is used in conjunction with **yaz_search()** to specify the preferred record syntax for retrieval.

yaz_wait (PHP 4 >= 4.0.1)

Executes queries

```
int yaz_wait (int id, string syntax)
```

This function carries out networked (blocked) activity for outstanding requests which have been prepared by the functions **yaz_connect()**, **yaz_search()**. **yaz_wait()** returns when all targets have either completed all requests or otherwise completed (in case of errors).

LXXVIII. YP/NIS Functions

NIS (formerly called Yellow Pages) allows network management of important administrative files (e.g. the password file). For more information refer to the NIS manpage and Introduction to YP/NIS (<http://www.desy.de/~sieversm/ypdoku/ypdoku/ypdoku.html>). There is also a book called Managing NFS and NIS (<http://www.oreilly.com/catalog/nfs/noframes.html>) by Hal Stern.

To get these functions to work, you have to configure PHP with `-with-yp`(PHP 3) or `-enable-yp`(PHP 4).

yp_get_default_domain (PHP 3>= 3.0.7, PHP 4)

Fetches the machine's default NIS domain

```
int yp_get_default_domain (void )
```

Yp_get_default_domain() returns the default domain of the node or FALSE. Can be used as the domain parameter for successive NIS calls.

A NIS domain can be described a group of NIS maps. Every host that needs to look up information binds itself to a certain domain. Refer to the documents mentioned at the beginning for more detailed information.

Example 1. Example for the default domain

```
<?php
$domain = yp_get_default_domain();
echo "Default NIS domain is: " . $domain;
?>
```

yp_order (PHP 3>= 3.0.7, PHP 4)

Returns the order number for a map

```
int yp_order (string domain, string map)
```

Yp_order() returns the order number for a map or FALSE.

Example 1. Example for the NIS order

```
<?php
$number = yp_order($domain,$mapname);
echo "Order number for this map is: " . $order;
?>
```

See also **yp-get-default-domain()**.

yp_master (PHP 3>= 3.0.7, PHP 4)

Returns the machine name of the master NIS server for a map

```
string yp_master (string domain, string map)
```

Yp_master() returns the machine name of the master NIS server for a map.

Example 1. Example for the NIS master

```
<?php
```

```
$number = yp_master ($domain, $mapname);
echo "Master for this map is: " . $master;
?>
```

See also **yp-get-default-domain()**.

yp_match (PHP 3>= 3.0.7, PHP 4)

Returns the matched line

```
string yp_match (string domain, string map, string key)
```

Yp_match() returns the value associated with the passed key out of the specified map or FALSE. This key must be exact.

Example 1. Example for NIS match

```
<?php
$entry = yp_match ($domain, "passwd.byname", "joe");
echo "Matched entry is: " . $entry;
?>
```

In this case this could be: joe:#joe:11111:100:Joe User:/home/j/joe:/usr/local/bin/bash

See also **yp-get-default-domain()**

yp_first (PHP 3>= 3.0.7, PHP 4)

Returns the first key-value pair from the named map

```
array yp_first (string domain, string map)
```

Yp_first() returns the first key-value pair from the named map in the named domain, otherwise FALSE.

Example 1. Example for the NIS first

```
<?php
$entry = yp_first($domain, "passwd.byname");
$key = key($entry);
echo "First entry in this map has key " . $key
. " and value " . $entry[$key];
?>
```

See also **yp-get-default-domain()**

yp_next (PHP 3>= 3.0.7, PHP 4)

Returns the next key-value pair in the named map.

```
array yp_next (string domain, string map, string key)
```

Yp_next() returns the next key-value pair in the named map after the specified key or FALSE.

Example 1. Example for NIS next

```
<?php
$entry = yp_next ($domain, "passwd.byname", "joe");

if (!$entry) {
    echo yp_errno() . ":" . yp_err_string();
}

$key = key ($entry);

echo "The next entry after joe has key " . $key
    . " and value " . $entry[$key];
?>
```

See also **yp-get-default-domain()**.

LXXIX. Zlib Compression Functions

This module uses the functions of zlib (<http://www.info-zip.org/pub/infozip/zlib/>) by Jean-loup Gailly and Mark Adler to transparently read and write gzip (.gz) compressed files. You have to use a zlib version >= 1.0.9 with this module.

This module contains versions of most of the [filesystem](#) functions which work with gzip-compressed files (and uncompressed files, too, but not with sockets).

Note: The current CVS version 4.0.4-dev introduces a fopen-wrapper for .gz-files, so that you can use a special 'zlib:' URL to access compressed files transparently using the normal f*() file access functions if you prepend the filename or path with a 'zlib:' prefix when calling **fopen()**.

This feature requires a C runtime library that provides the **fopencookie()** function. To my current knowledge the GNU libc is the only library that provides this feature.

Small code example

Opens a temporary file and writes a test string to it, then it prints out the content of this file twice.

Example 1. Small Zlib Example

```
<?php

$filename = tempnam ('/tmp', 'zlibtest').'.gz';
print "<html>\n<head></head>\n<body>\n<pre>\n";
$s = "Only a test, test, test, test, test, test, test, test!\n";

// open file for writing with maximum compression
$zp = gzopen($filename, "w9");

// write string to file
gzwrite($zp, $s);

// close file
gzclose($zp);

// open file for reading
$zp = gzopen($filename, "r");

// read 3 char
print gzread($zp, 3);

// output until end of the file and close it.
gzpassthru($zp);

print "\n";

// open file and print content (the 2nd time).
if (readgzfile($filename) != strlen($s)) {
    echo "Error with zlib functions!";
}
unlink($filename);
print "</pre>\n</h1></body>\n</html>\n";

?>
```


gzclose (PHP 3, PHP 4)

Close an open gz-file pointer

```
int gzclose (int zp)
```

The gz-file pointed to by *zp* is closed.

Returns true on success and false on failure.

The gz-file pointer must be valid, and must point to a file successfully opened by **gzopen()**.

gzeof (PHP 3, PHP 4)

Test for end-of-file on a gz-file pointer

```
int gzeof (int zp)
```

Returns true if the gz-file pointer is at EOF or an error occurs; otherwise returns false.

The gz-file pointer must be valid, and must point to a file successfully opened by **gzopen()**.

gzfile (PHP 3, PHP 4)

Read entire gz-file into an array

```
array gzfile (string filename [, int use_include_path])
```

Identical to **readgzfile()**, except that **gzfile()** returns the file in an array.

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 **readgzfile()**, and **gzopen()**.

gzgetc (PHP 3, PHP 4)

Get character from gz-file pointer

```
string gzgetc (int zp)
```

Returns a string containing a single (uncompressed) character read from the file pointed to by *zp*. Returns FALSE on EOF (as does **gzeof()**).

The gz-file pointer must be valid, and must point to a file successfully opened by **gzopen()**.

See also **gzopen()**, and **gzgets()**.

gzgets (PHP 3, PHP 4)

Get line from file pointer

```
string gzgets (int zp, int length)
```

Returns a (uncompressed) 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, or on EOF (whichever comes first).

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by **gzopen()**.

See also **gzopen()**, **gzgetc()**, and **fgets()**.

gzgetss (PHP 3, PHP 4)

Get line from gz-file pointer and strip HTML tags

```
string gzgetss (int zp, int length [, string allowable_tags])
```

Identical to **gzgets()**, except that **gzgetss()** 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 **gzgets()**, **gzopen()**, and **strip_tags()**.

gzopen (PHP 3, PHP 4)

Open gz-file

```
int gzopen (string filename, string mode [, int use_include_path])
```

Opens a gzip (.gz) file for reading or writing. The mode parameter is as in **fopen()** ("rb" or "wb") but can also include a compression level ("wb9") or a strategy: 'f' for filtered data as in "wb6f", 'h' for Huffman only compression as in "wb1h". (See the description of deflateInit2 in zlib.h for more information about the strategy parameter.)

Gzopen() can be used to read a file which is not in gzip format; in this case **gzread()** will directly read from the file without decompression.

Gzopen() returns a file pointer to the file opened, after that, everything you read from this file descriptor will be transparently decompressed and what you write gets compressed.

If the open fails, the function returns false.

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. Gzopen() Example

```
$fp = gzopen ("tmp/file.gz", "r");
```

See also **gzclose()**.

gzpassthru (PHP 3, PHP 4)

Output all remaining data on a gz-file pointer

```
int gzpassthru (int zp)
```

Reads to EOF on the given gz-file pointer and writes the (uncompressed) results to standard output.

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by **gzopen()**.

The gz-file is closed when **gzpassthru()** is done reading it (leaving *zp* useless).

gzputs (PHP 3, PHP 4)

Write to a gz-file pointer

```
int gzputs (int zp, string str [, int length])
```

Gzputs() is an alias to **gzwrite()**, and is identical in every way.

gzread (PHP 3, PHP 4)

Binary-safe gz-file read

```
string gzread (int zp, int length)
```

gzread() reads up to *length* bytes from the gz-file pointer referenced by *zp*. Reading stops when *length* (uncompressed) bytes have been read or EOF is reached, whichever comes first.

```
// get contents of a gz-file into a string
$filename = "/usr/local/something.txt.gz";
$zd = gzopen ($filename, "r");
$contents = gzread ($zd, 10000);
gzclose ($zd);
```

See also **gzwrite()**, **gzopen()**, **gzgets()**, **gzgetss()**, **gzfile()**, and **gzpassthru()**.

gzrewind (PHP 3, PHP 4)

Rewind the position of a gz-file pointer

```
int gzrewind (int zp)
```

Sets the file position indicator for *zp* 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 **gzopen()**.

See also **gzseek()** and **gztell()**.

gzseek (PHP 3, PHP 4)

Seek on a gz-file pointer

```
int gzseek (int zp, int offset)
```

Sets the file position indicator for the file referenced by *zp* to *offset* bytes into the file stream. Equivalent to calling (in C) `gzseek(zp, offset, SEEK_SET)`.

If the file is opened for reading, this function is emulated but can be extremely slow. If the file is opened for writing, only forward seeks are supported; **gzseek** then compresses a sequence of zeroes up to the new starting position.

Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

See also **gtell()** and **gzrewind()**.

gtell (PHP 3, PHP 4)

Tell gz-file pointer read/write position

```
int gtell (int zp)
```

Returns the position of the file pointer referenced by *zp*; 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 **gzopen()**.

See also **gzopen()**, **gzseek()** and **gzrewind()**.

gzwrite (PHP 3, PHP 4)

Binary-safe gz-file write

```
int gzwrite (int zp, string string [, int length])
```

Gzwrite() writes the contents of *string* to the gz-file stream pointed to by *zp*. If the *length* argument is given, writing will stop after *length* (uncompressed) 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 **gzread()**, **gzopen()**, and **gzputs()**.

readgzfile (PHP 3, PHP 4)

Output a gz-file

```
int readgzfile (string filename [, int use_include_path])
```

Reads a file, decompresses it and writes it to standard output.

Readgzfile() can be used to read a file which is not in gzip format; in this case **readgzfile()** will directly read from the file without decompression.

Returns the number of (uncompressed) bytes read from the file. If an error occurs, false is returned and unless the function was called as `@readgzfile`, an error message is printed.

The file `filename` 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 **gzpassthru()**, **gzfile()**, and **gzopen()**.

gzcompress (PHP 4 >= 4.0.1)

Deflate a string

```
string gzcompress (string data [, int level])
```

This function returns a compressed version of the input `data` using the deflate algorithm, or false if an error is encountered. The optional parameter `level` can be given as 0 for no compression up to 9 for maximum compression.

Note: This is *not* the same as gzip compression, which includes some header data. See **gzencode()** for gzip compression.

See also **gzuncompress()**.

gzuncompress (PHP 4 >= 4.0.1)

Uncompress a deflated string

```
string gzuncompress (string data [, int length])
```

This function takes `data` compressed by **gzcompress()** and returns the original uncompressed data or false on error. The function will return an error if the uncompressed data is more than 256 times the length of the compressed input `data` or more than the optional parameter `length`.

See also **gzcompress()**.

Part V. Appendixes

Appendix A. Migrating from older versions of PHP

Migrating from PHP 3 to PHP 4

Migration from PHP 3 to PHP 4 is relatively easy, and should not require you to change your code in any way. There are minor incompatibilities between the two versions; You may want to check the incompatibilities list to make sure that you're indeed not affected by them (the chances you're affected by these incompatibilities are extremely slim).

Running PHP 3 and PHP 4 concurrently

Recent operating systems provide the ability to perform versioning and scoping. This features make it possible to let PHP 3 and PHP 4 run as concurrent modules in one Apache server.

This feature is known to work on the following platforms:

- Linux with recent binutils (binutils 2.9.1.0.25 tested)
- Solaris 2.5 or better
- FreeBSD (3.2, 4.0 tested)

To enable it, configure PHP3 and PHP4 to use APXS (`-with-apxs`) and the necessary link extensions (`-enable-versioning`). Otherwise, all standard installations instructions apply. For example:

```
$ ./configure \
  -with-apxs=/apache/bin/apxs \
  -enable-versioning \
  -with-mysql \
  -enable-track-vars
```

Migrating Configuration Files

Global Configuration File

The global configuration file, `php3.ini`, has changed its name to `php.ini`.

Apache Configuration Files

The MIME types recognized by the PHP module have changed.

```
application/x-httdp-php3      ->      application/x-httdp-php
application/x-httdp-php3-source ->      application/x-httdp-php-source
```

You can make your configuration files work with both versions of PHP (depending on which one is currently compiled into the server), using the following syntax:

```
AddType application/x-httdp-php3 .php3
AddType application/x-httdp-php3-source .php3s

AddType application/x-httdp-php .php
AddType application/x-httdp-php-source .phps
```

In addition, the PHP directive names for Apache have changed.

Starting with PHP 4.0, there are only four Apache directives that relate to PHP:

```
php_value [PHP directive name] [value]
php_flag [PHP directive name] [On|Off]
```

```
php_admin_value [PHP directive name] [value]
php_admin_flag [PHP directive name] [On|Off]
```

There are two differences between the Admin values and the non admin values:

- Admin values (or flags) can only appear in the server-wide apache configuration files (e.g., httpd.conf).
- Standard values (or flags) cannot control certain PHP directives, for example - safe mode (if you could override safe mode settings in .htaccess files, it would defeat safe-mode's purpose). In contrast, Admin values can modify the value of any PHP directive.

To make the transition process easier, PHP 4.0 is bundled with scripts that automatically convert your Apache configuration and .htaccess files to work with both PHP 3.0 and PHP 4.0. These scripts do NOT convert the mime type lines! You have to convert these yourself.

To convert your Apache configuration files, run the apconf-conv.sh script (available in the scripts/apache/ directory). For example:

```
~/php4/scripts/apache:# ./apconf-conv.sh /usr/local/apache/conf/httpd.conf
```

Your original configuration file will be saved in httpd.conf.orig.

To convert your .htaccess files, run the aphtaccess-conv.sh script (available in the scripts/apache/ directory as well):

```
~/php4/scripts/apache:# find / -name .htaccess -exec ./aphtaccess-conv.sh {} \;
```

Likewise, your old .htaccess files will be saved with an .orig prefix.

The conversion scripts require awk to be installed.

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 ("" != $key) {
    echo "$key";
    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 its 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. Migrating from PHP 3.0 to PHP 4.0

What has changed in PHP 4.0

PHP 4.0 and the integrated Zend engine have greatly improved PHP's performance and capabilities, but great care has been taken to break as little existing code as possible. So migrating your code from PHP 3.0 to 4.0 should be much easier than migrating from PHP/INI 2.0 to PHP 3.0. A lot of existing PHP 3.0 code should be ready to run without changes, but you should still know about the few differences and take care to test your code before switching versions in production environments. The following should give you some hints about what to look for.

Parser behavior

Parsing and execution are now two completely separated steps, no execution of a file's code will happen until the complete file and everything it requires has completely and successfully been parsed.

One of the new requirements introduced with this split is that required and included files now have to be syntactically complete. You can no longer spread the different controlling parts of a control structure across file boundaries. That is you cannot start a `for` or `while` loop, an `if` statement or a `switch` block in one file and have the end of loop, `else`, `endif`, `case` or `break` statements in a different file.

It is still perfectly legal to include additional code within loops or other control structures, only the controlling keywords and corresponding curly braces `{ ... }` have to be within the same compile unit (file or `eval()`ed string).

This should not harm too much as spreading code like this should be considered as very bad style anyway.

Another thing no longer possible, though rarely seen in PHP 3.0 code is returning values from a required file. Returning a value from an included file is still possible.

Error reporting

Configuration changes

With PHP 3.0 the error reporting level was set as a simple numeric value formed by summing up the numbers related to different error levels. Usual values were 15 for reporting all errors and warnings or 7 for reporting everything but simple notice messages reporting bad style and things like that.

PHP 4.0 now has a greater set of different error and warning levels and comes with a configuration parser that now allows for symbolic constants to be used for setting up the intended behavior.

Error reporting level should now be configured by explicitly taking away the warning levels you do not want to generate error messages by x-oring them from the symbolic constant `E_ALL`. Sounds complicated? Well, let's say you want the error reporting system to report all but the simple style warnings that are categorized by the symbolic constant `E_NOTICE`. Then you'll put the following into your `php.ini`: `error_reporting = E_ALL & ~ (E_NOTICE)`. If you want to suppress warnings too you add up the appropriate constant within the braces using the binary or operator '|':
`error_reporting= E_ALL & ~ (E_NOTICE | E_WARNING).`

Warning

Using the old values 7 and 15 for setting up error reporting is a very bad idea as this suppresses some of the newly added error classes including parse errors. This may lead to very strange behavior as scripts might no longer work without error messages showing up anywhere.

This has led to a lot of unreproducible bug reports in the past where people reported script engine problems they were not capable to track down while the true case was usually some missing ')' in a required file that the parser was not able to report due to a misconfigured error reporting system.

So checking your error reporting setup should be the first thing to do whenever your scripts silently die. The Zend engine can be considered mature enough nowadays to not cause this kind of strange behavior.

Additional warning messages

A lot of existing PHP 3.0 code uses language constructs that should be considered as very bad style as this code, while doing the intended thing now, could easily be broken by changes in other places. PHP 4.0 will output a lot of notice messages in such situations where PHP 3.0 didn't. The easy fix is to just turn off E_NOTICE messages, but it is usually a good idea to fix the code instead.

The most common case that will now produce notice messages is the use of unquoted string constants as array indices. Both PHP 3.0 and 4.0 will fall back to interprete these as strings if no keyword or constant is known by that name, but whenever a constant by that name had been defined anywhere else in the code it might break your script. This can even become a security risk if some intruder manages to redefine string constants in a way that makes your script give him access rights he wasn't intended to have. So PHP 4.0 will now warn you whenever you use unquoted string constants as for example in `$HTTP_SERVER_VARS[REQUEST_METHOD]`. Changing it to `$HTTP_SERVER_VARS['REQUEST_METHOD']` will make the parser happy and greatly improve the style and security of your code.

Another thing PHP 4.0 will now tell you about is the use of uninitialized variables or array elements.

Initializers

Static variable and class member initializers only accept scalar values while in PHP 3.0 they accepted any valid expression. This is, once again, due to the split between parsing and execution as no code has yet been executed when the parser sees the initializer.

For classes you should use constructors to initialize member variables instead. For static variables anything but a simple static value rarely makes sense anyway.

`empty("0")`

The perhaps most controversial change in behavior has happened to the behavior of the `empty()`. A String containing only the character '0' (zero) is now considered empty while it wasn't in PHP 3.0.

This new behavior makes sense in web applications, with all input fields returning strings even if numeric input is requested, and with PHP's capabilities of automatic type conversion. But on the other hand it might break your code in a rather subtle way, leading to misbehavior that is hard to track down if you do not know about what to look for.

Missing functions

While PHP 4.0 comes with a lot of new features, functions and extensions, you may still find some functions from version 3.0 missing. A small number of core functions has vanished because they do not work with the new scheme of splitting parsing and execution as introduced into 4.0 with the Zend engine. Other functions and even complete extensions have become obsolete as newer functions and extensions serve the same task better and/or in a more general way. Some functions just simply haven't been ported yet and finally some functions or extensions may be missing due to license conflicts.

Functions missing due to conceptual changes

As PHP 4.0 now separates parsing from execution it is no longer possible to change the behavior of the parser (now embedded in the Zend engine) at runtime as parsing already happened by then. So the function `short_tags()` has ceased to exist. You can still change the parser's behavior by setting appropriate values in the `php.ini` file.

Another feature from PHP 3.0 that didn't make it into 4.0 is the experimental debugging interface as described in ??? in this manual. A separate true debugger is promised to be released as a Zend product but hasn't become visible yet.

Deprecate functions and extensions

The Adabas and Solid database extensions are no more. Long live the unified ODBC extension instead.

Changed status for unset()

`unset()`, although still available, is implemented a little different in PHP 4.0 so that it no longer counts as a 'real' function.

This has no direct consequences as the behavior of `unset()` didn't change, but testing for "unset" using `function_exists()` will return `false` as it would with other lowlevel functions like `echo()`.

Another more practical change is that it is no longer possible to call `unset()` indirectly, that is `$func="unset"; $func($somevar)` won't work anymore.

PHP 3.0 extension

Extensions written for PHP 3.0 will not work with PHP 4.0 anymore, neither as binaries nor at the source level. It is not too difficult to port your extensions to PHP 4.0 if you have access to the original sources. A detailed description of the actual porting process is not part of this text (yet).

Variable substitution in strings

PHP 4.0 adds a new mechanism to variable substitution in strings. You can now finally access object member variables and elements from multidimensional arrays within strings.

To do so you have to enclose your variables with curly braces with the dollar sign immediately following the opening brace: `{$...}`

To embed the value of an object member variable into a string you simply write `"text {$obj->member} text"` while in PHP 3.0 you had to use something like `"text ".$obj->member." text"`.

This should lead to more readable code, while it may break existing scripts written for PHP 3.0. But you can easily check for this kind of problem by checking for the character combination `${` in your code and by replacing it with `\${` with your favourite search-and-replace tool.

Cookies

PHP 3.0 had the bad habit of setting cookies in the reverse order of the `setcookie()` calls in your code. PHP 4.0 breaks with this habit and creates the cookie header lines in exactly the same order as you set the cookies in the code.

This might break some existing code, but the old behaviour was so strange to understand that it deserved a change to prevent further problems in the future.

Appendix C. PHP development

Adding functions to PHP 3

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 C-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 C-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 C-1. PHP Internal Types

IS_STRING	String
IS_DOUBLE	Double-precision floating point
IS_LONG	Long integer
IS_ARRAY	Array
IS_EMPTY	None
IS_USER_FUNCTION	??
IS_INTERNAL_FUNCTION	?? (if some of these cannot be passed to a function - delete)
IS_CLASS	??
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 becomes 0, 1 otherwise */
convert_string_to_number(arg1); /* Converts string to either LONG or DOUBLE 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", PHP 3 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)¹
- SET_VAR_DOUBLE(name,value)
- SET_VAR_LONG(name,value)

¹

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 C-3. Checking whether \$foo exists in a symbol table

```
if (hash_exists(active_symbol_table,"foo",sizeof("foo"))) { exists... }
else { doesn't exist }
```

Example C-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 C-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 C-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,estrndup(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 C-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 C-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 C-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 msq1.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 destructor 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 if it requires no destruction 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 PHP 3 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_struct` 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 `retval` `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.

E_COMPILE_ERROR

This is like an E_ERROR, except it is generated by the Zend Scripting Engine. Functions should not generate this type of error.

E_COMPILE_WARNING

This is like an E_WARNING, except it is generated by the Zend Scripting Engine. Functions should not generate this type of error.

E_USER_ERROR

This is like an E_ERROR, except it is generated in PHP code by using the PHP function `trigger_error()`. Functions should not generate this type of error.

E_USER_WARNING

This is like an E_WARNING, except it is generated by using the PHP function `trigger_error()`. Functions should not generate this type of error.

E_USER_NOTICE

This is like an E_NOTICE, except it is generated by using the PHP function `trigger_error()`. 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 D. The PHP Debugger

About the debugger

PHP 3 includes support for a network-based debugger.

PHP 4 does not yet have a similar debugging facility.

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 D-1. Debugger Line Types

Name	Meaning
<code>start</code>	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.

Name	Meaning
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 D-2. Debugger Error Types

Debugger	PHP Internal
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 D-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

```

