

PHP Manual

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Andrei Zmievski

Jouni Ahto

Edited by
Stig Sæther Bakken

Egon Schmid

27-04-2002

Copyright © 1997, 1998, 1999, 2000, 2001, 2002 by the PHP Documentation Group

Copyright

This manual is © Copyright 1997, 1998, 1999, 2000, 2001, 2002 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.

The 'Extending PHP 4.0' section of this manual is copyright © 2000 by Zend Technologies, Ltd. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

PHP Manual

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

by

Edited by Stig Sæther Bakken

Edited by Egon Schmid

Published 27-04-2002

Copyright © 1997, 1998, 1999, 2000, 2001, 2002 by the PHP Documentation Group

Copyright

This manual is © Copyright 1997, 1998, 1999, 2000, 2001, 2002 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.

The 'Extending PHP 4.0' section of this manual is copyright © 2000 by Zend Technologies, Ltd. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Table of Contents

Preface	i
I. Getting Started	1
1. Introduction	1
What is PHP?.....	2
What can PHP do?.....	2
2. Installation.....	5
General Installation Considerations	6
Unix/HP-UX installs	6
Unix/Linux installs	8
Using Packages	8
Unix/Mac OS X installs.....	8
Using Packages	8
Compiling for OS X server.....	8
Compiling for MacOS X client.....	10
Unix/OpenBSD installs	11
Using Ports.....	11
Using Packages	11
Unix/Solaris installs.....	11
Required software	12
Using Packages	12
Installation on UNIX systems	12
Apache Module Quick Reference.....	13
Building	14
Installation on Windows systems	14
Windows InstallShield	14
Manual Installation Steps.....	14
Building from source	17
Preparations	17
Putting it all together.....	17
Compiling.....	18
Installation of Windows extensions	19
Servers-CGI/Commandline	21
Testing.....	21
Benchmarking	21
Servers-Apache.....	22
Details of installing PHP with Apache on Unix	22
Installing PHP on Windows with Apache 1.3.x.....	25
Servers-Caudium	26
Servers-fhttpd	27
Servers-IIS/PWS.....	27
Windows and PWS/IIS 3	27
Windows and PWS 4 or newer	28
Windows NT/2000/XP and IIS 4 or newer.....	29
Servers-Netscape and iPlanet	30
Installing PHP with Netscape on Sun Solaris.....	30

Installing PHP with Netscape on Windows	32
Servers-OmniHTTPd Server	34
OmniHTTPd 2.0b1 and up for Windows	34
Servers-Oreilly Website Pro	34
Oreilly Website Pro 2.5 and up for Windows	34
Servers-Xitami	35
Xitami for Windows	35
Servers-Other web servers	35
Problems?	35
Read the FAQ	36
Other problems	36
Bug reports	36
Complete list of configure options	36
Configure Options in PHP 4	37
Database options	37
Graphics options	40
Misc options	41
PHP options	48
Server options	48
XML options	49
3. Configuration	51
The configuration file	52
General Configuration Directives	53
Safe Mode Configuration Directives	58
Debugger Configuration Directives	59
Extension Loading Directives	59
mSQL Configuration Directives	59
Postgres Configuration Directives	59
SESAM Configuration Directives	60
Sybase Configuration Directives	60
Sybase-CT Configuration Directives	61
Informix Configuration Directives	62
BC Math Configuration Directives	62
Browser Capability Configuration Directives	63
Multi-Byte String Configuration Directives	63
Exif Configuration Directives	63
4. Security	65
General considerations	66
Installed as CGI binary	66
Possible attacks	67
Case 1: only public files served	67
Case 2: using --enable-force-cgi-redirect	68
Case 3: setting doc_root or user_dir	68
Case 4: PHP parser outside of web tree	68
Installed as an Apache module	69
Filesystem Security	69
Database Security	72
Designing Databases	72

Connecting to Database	72
Encrypted Storage Model	73
SQL Injection.....	73
Avoiding techniques	76
Error Reporting.....	77
Using Register Globals.....	79
User Submitted Data.....	80
Hiding PHP.....	81
Keeping Current	82
II. Language Reference	83
5. Basic syntax	83
Escaping from HTML	84
Instruction separation	85
Comments.....	85
6. Types	87
Introduction	88
Booleans	88
Syntax	89
Converting to boolean.....	89
Integers	90
Syntax	90
Integer overflow	90
Converting to integer.....	91
From booleans.....	92
From floating point numbers	92
From strings.....	92
From other types.....	92
Floating point numbers.....	92
Strings.....	93
Syntax	93
Single quoted.....	93
Double quoted	94
Heredoc	95
Variable parsing.....	96
Simple syntax.....	96
Complex (curly) syntax	97
String access by character	97
Useful functions	98
String conversion	99
Arrays	99
Syntax	100
Specifying with array().....	100
Creating/modifying with square-bracket syntax	100
Useful functions.....	101
Array do's and don'ts.....	101
Why is <code>\$foo[bar]</code> wrong?	101
So why is it bad then?.....	102

Examples.....	102
Objects.....	106
Object Initialization	106
Resource	107
Freeing resources	107
NULL	107
Syntax	107
Type Juggling	108
Type Casting	108
7. Variables.....	111
Basics.....	112
Predefined variables.....	113
Variable scope.....	114
Variable variables	117
Variables from outside PHP.....	118
HTML Forms (GET and POST).....	118
IMAGE SUBMIT variable names	119
HTTP Cookies	119
Environment variables	120
Dots in incoming variable names.....	120
Determining variable types.....	121
8. Constants	122
Syntax	123
Predefined constants	124
9. Expressions	125
10. Operators.....	129
Operator Precedence.....	130
Arithmetic Operators	130
Assignment Operators	131
Bitwise Operators	131
Comparison Operators.....	132
Error Control Operators.....	133
Execution Operators	134
Incrementing/Decrementing Operators	134
Logical Operators	135
String Operators.....	136
11. Control Structures	137
if.....	138
else	138
elseif	139
Alternative syntax for control structures	139
while	140
do..while.....	141
for.....	142
foreach.....	144
break	146
continue.....	147
switch.....	148

declare	150
Ticks	151
return	152
require()	152
include()	153
require_once()	157
include_once()	157
12. Functions	158
User-defined functions	159
Function arguments	159
Making arguments be passed by reference	159
Default argument values	160
Variable-length argument lists	161
Returning values	161
old_function	162
Variable functions	163
13. Classes and Objects	164
class	165
extends	167
Constructors	168
::	171
parent	172
Serializing objects - objects in sessions	173
The magic functions __sleep and __wakeup	174
References inside the constructor	174
14. References Explained	178
What References Are	179
What References Do	179
What References Are Not	180
Passing by Reference	180
Returning References	181
Unsetting References	182
Spotting References	182
global References	182
\$this	183
III. Features	184
15. Error Handling	184
16. Creating and manipulating images	189
17. HTTP authentication with PHP	191
18. Cookies	194
19. Handling file uploads	196
POST method uploads	197
Common Pitfalls	199
Uploading multiple files	199
PUT method support	200
20. Using remote files	202
21. Connection handling	205

22. Persistent Database Connections	207
23. Safe Mode	210
Functions restricted/disabled by safe mode.....	212
IV. Function Reference	216
I. Apache-specific Functions	216
apache_child_terminate	217
apache_lookup_uri	217
apache_note	218
apache_setenv	218
ascii2ebcdic	218
ebcdic2ascii	218
getallheaders	219
virtual.....	219
II. Array Functions	221
array	223
array_change_key_case	224
array_chunk	225
array_count_values	226
array_diff	227
array_fill	228
array_filter	228
array_flip.....	230
array_intersect	231
array_key_exists	231
array_keys.....	232
array_map	233
array_merge	237
array_merge_recursive	238
array_multisort	239
array_pad	240
array_pop.....	241
array_push	241
array_rand.....	242
array_reduce	243
array_reverse	244
array_search.....	245
array_shift.....	245
array_slice.....	246
array_splice.....	247
array_sum	248
array_unique	249
array_unshift.....	250
array_values.....	251
array_walk	252
arsort	253
asort	254
compact.....	255

count	256
current.....	257
each.....	257
end	259
extract	259
in_array.....	261
key	263
krsort.....	263
ksort	264
list	265
natcasesort	266
natsort	267
next	268
pos.....	268
prev	268
range	269
reset.....	270
rsort.....	270
shuffle	271
sizeof.....	271
sort	272
uasort	273
uksort	273
usort	274
III. Aspell functions [deprecated]	278
aspell_check	279
aspell_check_raw.....	279
aspell_new	279
aspell_suggest.....	280
IV. BCMath Arbitrary Precision Mathematics Functions	281
bcadd.....	282
bccomp	282
bcdiv	282
bcmul	282
bcmul	282
bcpow.....	283
bcscale	283
bcsqrt	283
bcsub.....	283
V. Bzip2 Compression Functions	285
bzclose	287
bzcompress	287
bzdecompress	287
bzerrno.....	288
bzerror.....	288
bzerrstr.....	289
bzflush.....	289
bzopen.....	289

bzread	290
bzwrite	290
VI. Calendar functions	292
cal_days_in_month.....	294
cal_from_jd.....	294
cal_info	294
cal_to_jd	294
easter_date	294
easter_days	295
FrenchToJD	296
GregorianToJD	296
JDDayOfWeek.....	296
JDMonthName	297
JDToFrench	298
JDToGregorian	298
JDToJewish.....	298
JDToJulian	298
jdtounix.....	299
JewishToJD.....	299
JulianToJD.....	299
unixtojd.....	300
VII. CCVS API Functions	301
ccvs_add	302
ccvs_auth	302
ccvs_command	302
ccvs_count	302
ccvs_delete	303
ccvs_done	303
ccvs_init.....	303
ccvs_lookup.....	304
ccvs_new	304
ccvs_report	304
ccvs_return	305
ccvs_reverse.....	305
ccvs_sale.....	305
ccvs_status.....	305
ccvs_textvalue	306
ccvs_void.....	306
VIII. COM support functions for Windows	307
COM	310
VARIANT.....	311
com_addrf.....	312
com_get	312
com_invoke.....	313
com_isenum.....	313
com_load	313
com_load_typelib	313
com_propget.....	314

com_propput.....	314
com_propset	314
com_release	314
com_set.....	314
IX. Class/Object Functions	316
call_user_method.....	319
call_user_method_array	319
class_exists	320
get_class	320
get_class_methods.....	320
get_class_vars.....	322
get_declared_classes.....	323
get_object_vars	323
get_parent_class	325
is_a.....	325
is_subclass_of.....	325
method_exists	325
X. ClibPDF functions.....	327
cpdf_add_annotation	332
cpdf_add_outline	332
cpdf_arc	332
cpdf_begin_text	333
cpdf_circle	333
cpdf_clip	333
cpdf_close.....	334
cpdf_closepath.....	334
cpdf_closepath_fill_stroke.....	334
cpdf_closepath_stroke	334
cpdf_continue_text	335
cpdf_curveto	335
cpdf_end_text	335
cpdf_fill.....	336
cpdf_fill_stroke.....	336
cpdf_finalize	336
cpdf_finalize_page.....	336
cpdf_global_set_document_limits.....	337
cpdf_import_jpeg	337
cpdf_lineto.....	337
cpdf_moveto	338
cpdf_newpath.....	338
cpdf_open	338
cpdf_output_buffer	339
cpdf_page_init	339
cpdf_place_inline_image.....	339
cpdf_rect.....	340
cpdf_restore	340
cpdf_rlineto.....	340
cpdf_rmoveto.....	341

cpdf_rotate	341
cpdf_rotate_text	341
cpdf_save	342
cpdf_save_to_file	342
cpdf_scale	342
cpdf_set_action_url	342
cpdf_set_char_spacing	343
cpdf_set_creator	343
cpdf_set_current_page	343
cpdf_set_font	343
cpdf_set_font_directories	344
cpdf_set_font_map_file	344
cpdf_set_horiz_scaling	344
cpdf_set_keywords	344
cpdf_set_leading	345
cpdf_set_page_animation	345
cpdf_set_subject	345
cpdf_set_text_matrix	346
cpdf_set_text_pos	346
cpdf_set_text_rendering	346
cpdf_set_text_rise	346
cpdf_set_title	347
cpdf_set_viewer_preferences	347
cpdf_set_word_spacing	347
cpdf_setdash	347
cpdf_setflat	348
cpdf_setgray	348
cpdf_setgray_fill	348
cpdf_setgray_stroke	348
cpdf_setlinecap	348
cpdf_setlinejoin	349
cpdf_setlinewidth	349
cpdf_setmiterlimit	349
cpdf_setrgbcolor	349
cpdf_setrgbcolor_fill	350
cpdf_setrgbcolor_stroke	350
cpdf_show	350
cpdf_show_xy	350
cpdf_stringwidth	351
cpdf_stroke	351
cpdf_text	351
cpdf_translate	351
XI. Crack functions	353
crack_check	355
crack_closedict	355
crack_getlastmessage	355
crack_opendict	356
XII. CURL, Client URL Library Functions	357

curl_close.....	360
curl_errno	360
curl_error	360
curl_exec.....	360
curl_getinfo.....	361
curl_init.....	361
curl_setopt	362
curl_version	365
XIII. Cybercash payment functions	366
cybercash_base64_decode.....	367
cybercash_base64_encode.....	367
cybercash_decr	367
cybercash_encr	367
XIV. Crédit Mutuel CyberMUT functions	368
cybermut_creeformulairecm	369
cybermut_creeerreponsecm.....	369
cybermut_testmac	370
XV. Cyrus IMAP administration functions	372
cyrus_authenticate	373
cyrus_bind	373
cyrus_close	373
cyrus_connect.....	373
cyrus_query	374
cyrus_unbind	374
XVI. Character type functions	375
ctype_alnum	376
ctype_alpha.....	376
ctype_cntrl.....	376
ctype_digit.....	376
ctype_graph	376
ctype_lower	377
ctype_print.....	377
ctype_punct.....	377
ctype_space.....	377
ctype_upper	378
ctype_xdigit	378
XVII. Database (dbm-style) abstraction layer functions.....	379
dba_close	383
dba_delete.....	383
dba_exists	383
dba_fetch	383
dba_firstkey	384
dba_insert	384
dba_nextkey.....	384
dba_open.....	385
dba_optimize	385
dba_popen.....	385
dba_replace.....	386

dba_sync	386
XVIII. Date and Time functions	387
checkdate	388
date	388
getdate.....	391
gettimeofday	391
gmdate	392
gmmktime.....	392
gmstrftime.....	392
localtime	393
microtime.....	393
mktime	394
strftime.....	395
strtotime	398
time	399
XIX. dBase functions	400
dbase_add_record	401
dbase_close.....	401
dbase_create	401
dbase_delete_record	402
dbase_get_record.....	402
dbase_get_record_with_names.....	403
dbase_numfields	403
dbase_numrecords	403
dbase_open	404
dbase_pack	404
dbase_replace_record	404
XX. DBM Functions	405
dblist	406
dbmclose.....	406
dbmdelete	406
dbmexists.....	406
dbmfetch	406
dbmfirstkey	406
dbminsert	407
dbmnextkey	407
dbmopen	407
dbmreplace	408
XXI. dbx functions.....	409
dbx_close.....	410
dbx_compare	410
dbx_connect.....	411
dbx_error	412
dbx_query	413
dbx_sort	416
XXII. DB++ Functions	418
dbplus_add.....	422
dbplus_aql.....	422

dbplus_chdir	422
dbplus_close	423
dbplus_curr	423
dbplus_errcode	423
dbplus_errno	424
dbplus_find	424
dbplus_first	425
dbplus_flush	425
dbplus_freealllocks	425
dbplus_freelock	426
dbplus_freerlocks	426
dbplus_getlock	427
dbplus_getunique	427
dbplus_info	427
dbplus_last	428
dbplus_lockrel	428
dbplus_next	428
dbplus_open	429
dbplus_prev	429
dbplus_rchperm	430
dbplus_rcreate	430
dbplus_rctxexact	431
dbplus_rctlke	431
dbplus_resolve	431
dbplus_restorepos	432
dbplus_rkeys	432
dbplus_ropen	432
dbplus_rquery	433
dbplus_rrename	433
dbplus_rsecindex	434
dbplus_runlink	434
dbplus_rzap	434
dbplus_savepos	435
dbplus_setindex	435
dbplus_setindexbynumber	435
dbplus_sql	435
dbplus_tcl	436
dbplus_tremove	436
dbplus_undo	437
dbplus_undoprepere	437
dbplus_unlockrel	437
dbplus_unselect	438
dbplus_update	438
dbplus_xlockrel	438
dbplus_xunlockrel	439
XXIII. Direct IO functions	440
dio_close	441
dio_fcntl	441

dio_open	441
dio_read	442
dio_seek	442
dio_stat	443
dio_truncate	443
dio_write	443
XXIV. Directory functions	445
chdir	446
chroot	446
dir	446
closedir	447
getcwd	447
opendir	447
readdir	448
rewinddir	449
XXV. DOM XML functions	450
DomAttribute->name	457
DomAttribute->specified	457
DomAttribute->value	457
DomDocument->add_root [deprecated]	457
DomDocument->create_attribute	458
DomDocument->create_cdata_section	458
DomDocument->create_comment	458
DomDocument->create_element	459
DomDocument->create_entity_reference	459
DomDocument->create_processing_instruction	459
DomDocument->create_text_node	460
DomDocument->doctype	460
DomDocument->document_element	460
DomDocument->dump_file	461
DomDocument->dump_mem	462
DomDocument->get_element_by_id	463
DomDocument->get_elements_by_tagname	463
DomDocument->html_dump_mem	463
DomDocumentType->entities	464
DomDocumentType->internal_subset	464
DomDocumentType->name	464
DomDocumentType->notations	465
DomDocumentType->public_id	465
DomDocumentType->system_id	466
DomElement->get_attribute	466
DomElement->get_attribute_node	466
DomElement->get_elements_by_tagname	467
DomElement->has_attribute	467
DomElement->remove_attribute	467
DomElement->set_attribute	468
DomElement->set_attribute_node	468
DomElement->>tagname	468

DomNode->append_child	469
DomNode->append_sibling	470
DomNode->attributes	471
DomNode->child-nodes	471
DomNode->clone_node	471
DomNode->dump_node	471
DomNode->first_child	472
DomNode->get_content	472
DomNode->has_attributess	472
DomNode->has_child_nodes	472
DomNode->insert_before	473
DomNode->is_blank_node	474
DomNode->last_child	474
DomNode->next_sibling	474
DomNode->node_name	475
DomNode->node_type	475
DomNode->node_value	476
DomNode->owner_document	476
DomNode->parent_node	477
DomNode->prefix	477
DomNode->previous_sibling	478
DomNode->remove_child	478
DomNode->replace_child	479
DomNode->replace_node	479
DomNode->set_content	479
DomNode->set_name	480
DomNode->unlink_node	480
DomProcessingInstruction->data	480
DomProcessingInstruction->target	480
domxml_new_doc	481
domxml_open_file	481
domxml_open_mem	482
domxml_version	482
domxml_xmltree	482
xpath_eval	483
xpath_eval_expression	483
xpath_new_context	483
xptr_eval	484
xptr_new_context	484
XXVI. .NET functions	485
dotnet_load	486
XXVII. Error Handling and Logging Functions	487
error_log	488
error_reporting	489
restore_error_handler	490
set_error_handler	490
trigger_error	493
user_error	494

XXVIII. FrontBase Functions	495
fbsql_affected_rows	496
fbsql_autocommit	496
fbsql_change_user	496
fbsql_close	497
fbsql_commit	497
fbsql_connect	497
fbsql_create_blob	498
fbsql_create_clob	499
fbsql_create_db	499
fbsql_data_seek	500
fbsql_database	501
fbsql_database_password	501
fbsql_db_query	502
fbsql_db_status	502
fbsql_drop_db	503
fbsql_errno	503
fbsql_error	504
fbsql_fetch_array	504
fbsql_fetch_assoc	505
fbsql_fetch_field	506
fbsql_fetch_lengths	507
fbsql_fetch_object	507
fbsql_fetch_row	508
fbsql_field_flags	509
fbsql_field_len	509
fbsql_field_name	509
fbsql_field_seek	510
fbsql_field_table	510
fbsql_field_type	510
fbsql_free_result	511
fbsql_get_autostart_info	511
fbsql_hostname	512
fbsql_insert_id	512
fbsql_list_dbs	512
fbsql_list_fields	513
fbsql_list_tables	514
fbsql_next_result	514
fbsql_num_fields	515
fbsql_num_rows	515
fbsql_password	516
fbsql_pconnect	516
fbsql_query	517
fbsql_read_blob	518
fbsql_read_clob	519
fbsql_result	519
fbsql_rollback	520
fbsql_select_db	520

fbsql_set_lob_mode.....	521
fbsql_set_transaction.....	521
fbsql_start_db.....	521
fbsql_stop_db.....	522
fbsql_tablename.....	522
fbsql_username.....	523
fbsql_warnings.....	523
XXIX. filePro functions.....	524
filepro.....	525
filepro_fieldcount.....	525
filepro_fieldname.....	525
filepro_fieldtype.....	525
filepro_fieldwidth.....	525
filepro_retrieve.....	526
filepro_rowcount.....	526
XXX. Filesystem functions.....	527
basename.....	528
chgrp.....	528
chmod.....	528
chown.....	529
clearstatcache.....	529
copy.....	530
delete.....	530
dirname.....	531
disk_free_space.....	531
disk_total_space.....	532
diskfreespace.....	532
fclose.....	532
feof.....	533
fflush.....	533
fgetc.....	533
fgetcsv.....	533
fgets.....	534
fgetss.....	535
file.....	535
file_exists.....	536
file_get_contents.....	537
file_get_wrapper_data.....	537
file_register_wrapper.....	537
fileatime.....	539
filectime.....	539
filegroup.....	540
fileinode.....	540
filemtime.....	540
fileowner.....	541
fileperms.....	541
filesize.....	541
filetype.....	542

flock	542
fopen	543
fpassthru	545
fputs	545
fread	545
fscanf	546
fseek	547
fstat	547
ftell.....	548
ftruncate.....	548
fwrite.....	548
is_dir	549
is_executable	549
is_file	550
is_link	550
is_readable.....	550
is_uploaded_file.....	551
is_writable	551
is_writeable.....	552
link	552
linkinfo	552
lstat	553
mkdir.....	553
move_uploaded_file.....	554
parse_ini_file	555
pathinfo.....	556
pclose.....	557
popen	557
readfile	558
readlink	559
realpath	559
rename	560
rewind	560
rmdir	560
set_file_buffer.....	560
stat.....	561
symlink	562
tempnam	562
tmpfile.....	563
touch	564
umask.....	564
unlink	564
XXXI. Forms Data Format functions.....	566
fdf_add_template.....	568
fdf_close	568
fdf_create.....	568
fdf_get_file	569
fdf_get_status	569

fdf_get_value	569
fdf_next_field_name	569
fdf_open	570
fdf_save	570
fdf_set_ap	570
fdf_set_encoding	571
fdf_set_file	571
fdf_set_flags	571
fdf_set_javascript_action	571
fdf_set_opt	572
fdf_set_status	572
fdf_set_submit_form_action	572
fdf_set_value	572
XXXII. FriBiDi functions	574
fribidi_log2vis	575
XXXIII. FTP functions	576
ftp_cdup	578
ftp_chdir	578
ftp_close	578
ftp_connect	578
ftp_delete	579
ftp_exec	579
ftp_fget	579
ftp_fput	579
ftp_get	580
ftp_get_option	580
ftp_login	581
ftp_mdtm	581
ftp_mkdir	581
ftp_nlist	581
ftp_pasv	582
ftp_put	582
ftp_pwd	582
ftp_quit	582
ftp_rawlist	583
ftp_rename	583
ftp_rmdir	583
ftp_set_option	583
ftp_site	584
ftp_size	584
ftp_systype	585
XXXIV. Function Handling functions	586
call_user_func	587
call_user_func_array	587
create_function	588
func_get_arg	590
func_get_args	591
func_num_args	592

function_exists.....	592
get_defined_functions.....	593
register_shutdown_function	594
register_tick_function.....	594
unregister_tick_function.....	594
XXXV. Gettext	596
bind_textdomain_codeset.....	597
bindtextdomain	597
dcgettext	597
dcngettext	597
dgettext	598
dngettext	598
gettext	598
ngettext	599
textdomain	599
XXXVI. GMP functions	600
gmp_abs.....	601
gmp_add	601
gmp_and	601
gmp_clrbit.....	601
gmp_cmp	601
gmp_com	601
gmp_div	602
gmp_div_q.....	602
gmp_div_qr.....	602
gmp_div_r.....	603
gmp_divexact.....	603
gmp_fact	603
gmp_gcd	604
gmp_gcdext	604
gmp_hamdist	604
gmp_init.....	604
gmp_intval	605
gmp_invert.....	605
gmp_jacobi	605
gmp_legendre	606
gmp_mod.....	606
gmp_mul.....	606
gmp_neg	606
gmp_or.....	606
gmp_perfect_square	607
gmp_popcount.....	607
gmp_pow	607
gmp_powm	607
gmp_prob_prime	607
gmp_random.....	608
gmp_scan0.....	608
gmp_scan1	608

gmp_setbit	608
gmp_sign	609
gmp_sqrt	609
gmp_sqrtrm	609
gmp_strval	609
gmp_sub	610
gmp_xor	610
XXXVII. HTTP functions	611
header	612
headers_sent	614
setcookie	615
XXXVIII. Hyperwave functions	617
hw_Array2Objrec	622
hw_changeobject	622
hw_Children	622
hw_ChildrenObj	622
hw_Close	622
hw_Connect	623
hw_connection_info	623
hw_Cp	623
hw_Deleteobject	624
hw_DocByAnchor	624
hw_DocByAnchorObj	624
hw_Document_Attributes	624
hw_Document_BodyTag	625
hw_Document_Content	625
hw_Document_SetContent	625
hw_Document_Size	625
hw_dummy	626
hw_EditText	626
hw_Error	626
hw_ErrorMsg	626
hw_Free_Document	627
hw_GetAnchors	627
hw_GetAnchorsObj	627
hw_GetAndLock	627
hw_GetChildColl	628
hw_GetChildCollObj	628
hw_GetChildDocColl	628
hw_GetChildDocCollObj	628
hw_GetObject	628
hw_GetObjectByQuery	629
hw_GetObjectByQueryColl	629
hw_GetObjectByQueryCollObj	630
hw_GetObjectByQueryObj	630
hw_GetParents	630
hw_GetParentsObj	631
hw_getrellink	631

hw_GetRemote	631
hw_GetRemoteChildren	631
hw_GetSrcByDestObj	632
hw_GetText	632
hw_getusername	633
hw_Identify.....	633
hw_InCollections.....	633
hw_Info.....	634
hw_InsColl	634
hw_InsDoc.....	634
hw_insertanchors.....	634
hw_InsertDocument	635
hw_InsertObject	635
hw_mapid	635
hw_Modifyobject	636
hw_Mv.....	638
hw_New_Document	638
hw_Objrec2Array	639
hw_Output_Document	639
hw_pConnect.....	639
hw_PipeDocument	640
hw_Root	640
hw_setlinkroot.....	640
hw_stat.....	640
hw_Unlock	641
hw_Who	641
XXXIX. Hyperwave API functions	642
hw_api_attribute	644
hw_api_attribute->key.....	644
hw_api_attribute->langdepvalue	644
hw_api_attribute->value	644
hw_api_attribute->values	644
hw_api->checkin	645
hw_api->checkout	645
hw_api->children.....	646
hw_api->content.....	646
hw_api_content->mimetype	646
hw_api_content->read	647
hw_api->copy	647
hw_api->dbstat	647
hw_api->dcstat	647
hw_api->dstanchors.....	648
hw_api->dstofsrcanchors.....	648
hw_api_error->count	648
hw_api_error->reason.....	648
hw_api->find.....	649
hw_api->ftstat.....	649
hwapi_hgcsp	649

hw_api->hwstat	649
hw_api->identify	649
hw_api->info	650
hw_api->insert.....	650
hw_api->insertanchor	651
hw_api->insertcollection	651
hw_api->insertdocument	651
hw_api->link.....	652
hw_api->lock.....	652
hw_api->move.....	652
hw_api_content.....	652
hw_api->object.....	653
hw_api_object->assign	654
hw_api_object->attreditable	654
hw_api_object->count	654
hw_api_object->insert	654
hw_api_object.....	655
hw_api_object->remove	655
hw_api_object->title.....	655
hw_api_object->value	655
hw_api->objectbyanchor	655
hw_api->parents	656
hw_api_reason->description.....	656
hw_api_reason->type	656
hw_api->remove.....	656
hw_api->replace	657
hw_api->setcommittedversion.....	658
hw_api->srcanchors.....	658
hw_api->srcsofdst.....	658
hw_api->unlock.....	658
hw_api->user	659
hw_api->userlist	659
XL. ICAP Functions [deprecated]	660
icap_close	661
icap_create_calendar	661
icap_delete_calendar	661
icap_delete_event	661
icap_fetch_event.....	661
icap_list_alarms.....	662
icap_list_events	663
icap_open.....	663
icap_rename_calendar	663
icap_reopen.....	664
icap_snooze	664
icap_store_event	664
XLI. iconv functions	666
iconv	667
iconv_get_encoding	667

iconv_set_encoding	667
ob_iconv_handler	668
XLII. Image functions	669
exif_imagetype	670
exif_read_data	670
exif_thumbnail	673
getimagesize	673
image2wbmp	675
imagealphablending	675
imagearc	676
imagechar	676
imagecharup	676
imagecolorallocate	676
imagecolorat	677
imagecolorclosest	677
imagecolorclosestalpha	677
imagecolorclosestwb	678
imagecolordeallocate	678
imagecolorexact	678
imagecolorexactalpha	679
imagecolorresolve	679
imagecolorresolvealpha	679
imagecolorset	679
imagecolorsforindex	680
imagecolorstotal	680
imagecolortransparent	680
imagecopy	681
imagecopymerge	681
imagecopymergegray	681
imagecopyresampled	682
imagecopyresized	682
imagecreate	682
imagecreatefromgd	683
imagecreatefromgd2	683
imagecreatefromgd2part	683
imagecreatefromgif	684
imagecreatefromjpeg	685
imagecreatefrompng	685
imagecreatefromstring	686
imagecreatefromwbmp	686
imagecreatefromxbm	687
imagecreatefromxpm	687
imagecreatetruecolor	687
imagedashedline	688
imagedestroy	688
imageellipse	688
imagefill	688
imagefilledarc	689

imagefilledellipse.....	689
imagefilledpolygon.....	689
imagefilledrectangle.....	690
imagefilltoborder.....	690
imagefontheight.....	690
imagefontwidth.....	690
imageftbbox.....	691
imagefttext.....	691
imagegammacorrect.....	691
imagegd.....	691
imagegd2.....	692
imagegif.....	692
imageinterlace.....	693
imagejpeg.....	694
imageline.....	694
imageloadfont.....	694
imagepalettecopy.....	695
imagepng.....	695
imagepolygon.....	696
imagepsbbox.....	696
ImagePSCopyFont.....	697
imagepsencodefont.....	697
imagepsextendfont.....	697
imagepsfreefont.....	698
imagepsloadfont.....	698
imagepslantfont.....	698
imagepstext.....	698
imagerectangle.....	699
imagesetbrush.....	700
imagesetpixel.....	700
imagesetstyle.....	700
imagesethickness.....	701
imagesettile.....	701
imagestring.....	702
imagestringup.....	702
imagesx.....	702
imagesy.....	703
imagetruecolortopalette.....	703
imageftbbox.....	703
imagefttext.....	704
imagetypes.....	705
imagewbmp.....	706
iptcembed.....	706
jpeg2wbmp.....	707
png2wbmp.....	707
read_exif_data.....	707
XLIII. IMAP, POP3 and NNTP functions.....	708
imap_8bit.....	709

imap_alerts	709
imap_append.....	709
imap_base64.....	710
imap_binary.....	710
imap_body.....	710
imap_bodystruct	711
imap_check.....	711
imap_clearflag_full.....	711
imap_close.....	712
imap_createmailbox	712
imap_delete.....	713
imap_deletemailbox	714
imap_errors.....	714
imap_expunge.....	715
imap_fetch_overview	715
imap_fetchbody	716
imap_fetchheader	717
imap_fetchstructure	717
imap_get_quota	719
imap_getmailboxes.....	719
imap_getsubscribed	720
imap_header	721
imap_headerinfo.....	721
imap_headers.....	723
imap_last_error.....	723
imap_listmailbox	723
imap_listsubscribed	724
imap_mail.....	724
imap_mail_compose.....	724
imap_mail_copy.....	725
imap_mail_move.....	726
imap_mailboxmsginfo.....	726
imap_mime_header_decode.....	727
imap_msgno	728
imap_num_msg	728
imap_num_recent.....	728
imap_open	729
imap_ping.....	731
imap_popen	731
imap_qprint.....	731
imap_renamemailbox	732
imap_reopen	732
imap_rfc822_parse_adrlist	732
imap_rfc822_parse_headers.....	733
imap_rfc822_write_address	733
imap_scanmailbox.....	734
imap_search.....	734
imap_set_quota.....	735

imap_setacl	736
imap_setflag_full	736
imap_sort	737
imap_status	737
imap_subscribe	738
imap_thread	738
imap_uid	739
imap_undelete	739
imap_unsubscribe	739
imap_utf7_decode	740
imap_utf7_encode	740
imap_utf8	740
XLIV. Informix functions	741
ifx_affected_rows	743
ifx_blobinfile_mode	743
ifx_byteasvarchar	743
ifx_close	744
ifx_connect	744
ifx_copy_blob	745
ifx_create_blob	745
ifx_create_char	745
ifx_do	746
ifx_error	746
ifx_errormsg	746
ifx_fetch_row	747
ifx_fieldproperties	748
ifx_fielddtypes	748
ifx_free_blob	749
ifx_free_char	749
ifx_free_result	749
ifx_get_blob	749
ifx_get_char	750
ifx_getsqlca	750
ifx_htmltbl_result	751
ifx_nullformat	751
ifx_num_fields	751
ifx_num_rows	752
ifx_pconnect	752
ifx_prepare	752
ifx_query	753
ifx_textasvarchar	754
ifx_update_blob	755
ifx_update_char	755
ifxus_close_slob	755
ifxus_create_slob	755
ifxus_free_slob	755
ifxus_open_slob	756
ifxus_read_slob	756

ifxus_seek_slob	756
ifxus_tell_slob	756
ifxus_write_slob	757
XLV. InterBase functions	758
ibase_blob_add	759
ibase_blob_cancel	759
ibase_blob_close	759
ibase_blob_create	759
ibase_blob_echo	760
ibase_blob_get	760
ibase_blob_import	760
ibase_blob_info	761
ibase_blob_open	761
ibase_close	761
ibase_commit	761
ibase_connect	762
ibase_errmsg	763
ibase_execute	763
ibase_fetch_object	764
ibase_fetch_row	764
ibase_field_info	764
ibase_free_query	765
ibase_free_result	765
ibase_num_fields	765
ibase_pconnect	766
ibase_prepare	767
ibase_query	767
ibase_rollback	767
ibase_timefmt	767
ibase_trans	768
XLVI. Ingres II functions	769
ingres_autocommit	770
ingres_close	770
ingres_commit	770
ingres_connect	771
ingres_fetch_array	772
ingres_fetch_object	773
ingres_fetch_row	774
ingres_field_length	775
ingres_field_name	775
ingres_field_nullable	775
ingres_field_precision	776
ingres_field_scale	776
ingres_field_type	777
ingres_num_fields	777
ingres_num_rows	778
ingres_pconnect	778
ingres_query	779

ingres_rollback	780
XLVII. IRC Gateway Functions.....	781
ircg_channel_mode.....	782
ircg_disconnect.....	782
ircg_fetch_error_msg	782
ircg_get_username.....	782
ircg_html_encode	783
ircg_ignore_add.....	783
ircg_ignore_del.....	783
ircg_is_conn_alive.....	783
ircg_join.....	784
ircg_kick	784
ircg_lookup_format_messages	784
ircg_msg	784
ircg_nick	785
ircg_nickname_escape.....	785
ircg_nickname_unescape.....	785
ircg_notice	785
ircg_part.....	785
ircg_pconnect	786
ircg_register_format_messages	786
ircg_set_current	788
ircg_set_file	788
ircg_set_on_die.....	788
ircg_topic.....	788
ircg_whois	789
XLVIII. Java.....	790
java_last_exception_clear.....	793
java_last_exception_get.....	793
XLIX. LDAP functions.....	794
ldap_8859_to_t61	797
ldap_add	797
ldap_bind	798
ldap_close	798
ldap_compare	798
ldap_connect.....	800
ldap_count_entries.....	800
ldap_delete.....	800
ldap_dn2ufn	800
ldap_err2str.....	801
ldap_errno.....	801
ldap_error	802
ldap_explode_dn.....	802
ldap_first_attribute.....	803
ldap_first_entry.....	803
ldap_first_reference	803
ldap_free_result	804
ldap_get_attributes.....	804

ldap_get_dn	805
ldap_get_entries.....	805
ldap_get_option	806
ldap_get_values	807
ldap_get_values_len	807
ldap_list	808
ldap_mod_add	809
ldap_mod_del	809
ldap_mod_replace.....	809
ldap_modify.....	809
ldap_next_attribute	810
ldap_next_entry	810
ldap_next_reference.....	810
ldap_parse_reference	811
ldap_parse_result.....	811
ldap_read	811
ldap_rename	812
ldap_search	812
ldap_set_option.....	814
ldap_set_rebind_proc	815
ldap_sort	815
ldap_start_tls.....	816
ldap_t61_to_8859.....	816
ldap_unbind	816
L. Mail functions.....	818
ezmlm_hash.....	819
mail	819
L.I. mailparse functions	822
mailparse_determine_best_xfer_encoding	823
mailparse_msg_create	823
mailparse_msg_extract_part.....	823
mailparse_msg_extract_part_file.....	824
mailparse_msg_free.....	824
mailparse_msg_get_part.....	825
mailparse_msg_get_part_data	825
mailparse_msg_get_structure.....	826
mailparse_msg_parse	826
mailparse_msg_parse_file	826
mailparse_rfc822_parse_addresses	827
mailparse_stream_encode.....	827
mailparse_uudecode_all	828
L.II. Mathematical Functions.....	829
abs.....	830
acos	830
acosh	830
asin.....	830
asinh.....	831
atan	831

atan2	831
atanh	831
base_convert	832
bindec	832
ceil	832
cos	833
cosh	833
decbin	833
dechex	834
decoct	834
deg2rad	834
exp	834
expm1	835
floor	835
getrandmax	835
hexdec	836
hypot	836
is_finite	837
is_infinite	837
is_nan	837
lcg_value	837
log	838
log10	838
log1p	838
max	838
min	839
mt_getrandmax	839
mt_rand	839
mt_srand	840
number_format	840
octdec	842
pi	842
pow	842
rad2deg	843
rand	843
round	844
sin	844
sinh	845
sqrt	845
srand	845
tan	845
tanh	846
LIII. Multi-Byte String Functions	847
mb_convert_encoding	855
mb_convert_kana	855
mb_convert_variables	856
mb_decode_mimeheader	857
mb_decode_numericentity	857

mb_detect_encoding.....	858
mb_detect_order.....	858
mb_encode_mimeheader.....	860
mb_encode_numericentity.....	860
mb_ereg.....	861
mb_ereg_match.....	862
mb_ereg_replace.....	862
mb_ereg_search.....	863
mb_ereg_search_getpos.....	864
mb_ereg_search_getregs.....	864
mb_ereg_search_init.....	865
mb_ereg_search_pos.....	865
mb_ereg_search_regs.....	866
mb_ereg_search_setpos.....	866
mb_eregi.....	867
mb_eregi_replace.....	868
mb_get_info.....	868
mb_http_input.....	868
mb_http_output.....	869
mb_internal_encoding.....	869
mb_language.....	870
mb_output_handler.....	870
mb_parse_str.....	871
mb_preferred_mime_name.....	871
mb_regex_encoding.....	872
mb_send_mail.....	872
mb_split.....	873
mb_strcut.....	873
mb_strimwidth.....	874
mb_strlen.....	874
mb_strpos.....	874
mb_strrpos.....	875
mb_strwidth.....	875
mb_substitute_character.....	876
mb_substr.....	877
LIV. MCAL functions.....	878
mcal_append_event.....	880
mcal_close.....	880
mcal_create_calendar.....	880
mcal_date_compare.....	880
mcal_date_valid.....	880
mcal_day_of_week.....	880
mcal_day_of_year.....	881
mcal_days_in_month.....	881
mcal_delete_calendar.....	881
mcal_delete_event.....	881
mcal_event_add_attribute.....	882
mcal_event_init.....	882

mcal_event_set_alarm	882
mcal_event_set_category.....	882
mcal_event_set_class.....	882
mcal_event_set_description.....	883
mcal_event_set_end.....	883
mcal_event_set_recur_daily	883
mcal_event_set_recur_monthly_mday	883
mcal_event_set_recur_monthly_wday	884
mcal_event_set_recur_none	884
mcal_event_set_recur_weekly.....	884
mcal_event_set_recur_yearly	884
mcal_event_set_start	884
mcal_event_set_title	885
mcal_expunge.....	885
mcal_fetch_current_stream_event.....	885
mcal_fetch_event.....	886
mcal_is_leap_year	887
mcal_list_alarms.....	887
mcal_list_events	888
mcal_next_recurrence.....	888
mcal_open.....	888
mcal_popen.....	888
mcal_rename_calendar	889
mcal_reopen	889
mcal_snooze	889
mcal_store_event	889
mcal_time_valid	889
mcal_week_of_year.....	890
LV. Mcrypt Encryption Functions	891
mccrypt_cbc	896
mccrypt_cfb.....	896
mccrypt_create_iv	896
mccrypt_decrypt.....	897
mccrypt_ecb	898
mccrypt_enc_get_algorithms_name.....	898
mccrypt_enc_get_block_size	898
mccrypt_enc_get_iv_size	898
mccrypt_enc_get_key_size	899
mccrypt_enc_get_modes_name	899
mccrypt_enc_get_supported_key_sizes	899
mccrypt_enc_is_block_algorithm	899
mccrypt_enc_is_block_algorithm_mode	899
mccrypt_enc_is_block_mode.....	900
mccrypt_enc_self_test.....	900
mccrypt_encrypt.....	900
mccrypt_generic	901
mccrypt_generic_deinit.....	901
mccrypt_generic_end	902

mcrypt_generic_init.....	902
mcrypt_get_block_size.....	902
mcrypt_get_cipher_name.....	903
mcrypt_get_iv_size.....	903
mcrypt_get_key_size.....	904
mcrypt_list_algorithms.....	904
mcrypt_list_modes.....	904
mcrypt_module_close.....	905
mcrypt_module_get_algo_block_size.....	905
mcrypt_module_get_algo_key_size.....	906
mcrypt_module_get_supported_key_sizes.....	906
mcrypt_module_is_block_algorithm.....	906
mcrypt_module_is_block_algorithm_mode.....	906
mcrypt_module_is_block_mode.....	907
mcrypt_module_open.....	907
mcrypt_module_self_test.....	908
mcrypt_ofb.....	908
mdecrypt_generic.....	908
LVI. Mhash Functions.....	910
mhash.....	912
mhash_count.....	912
mhash_get_block_size.....	912
mhash_get_hash_name.....	913
mhash_keygen_s2k.....	913
LVII. Microsoft SQL Server functions.....	914
mssql_bind.....	915
mssql_close.....	915
mssql_connect.....	915
mssql_data_seek.....	916
mssql_execute.....	916
mssql_fetch_array.....	916
mssql_fetch_assoc.....	916
mssql_fetch_batch.....	917
mssql_fetch_field.....	917
mssql_fetch_object.....	918
mssql_fetch_row.....	918
mssql_field_length.....	918
mssql_field_name.....	918
mssql_field_seek.....	919
mssql_field_type.....	919
mssql_free_result.....	919
mssql_get_last_message.....	919
mssql_guid_string.....	919
mssql_init.....	920
mssql_min_error_severity.....	920
mssql_min_message_severity.....	920
mssql_next_result.....	920
mssql_num_fields.....	921

mssql_num_rows	921
mssql_pconnect	921
mssql_query	922
mssql_result	922
mssql_rows_affected	923
mssql_select_db	923
LVIII. Ming functions for Flash	924
ming_setcubicthreshold	926
ming_setscale	926
ming_useswfversion	926
SWFAction	926
SWFBitmap	937
SWFBitmap->getHeight	939
SWFBitmap->getWidth	939
SWFbutton	940
swfbutton_keypress	943
SWFbutton->addAction	943
SWFbutton->addShape	943
SWFbutton->setAction	944
SWFbutton->setdown	944
SWFbutton->setHit	944
SWFbutton->setOver	945
SWFbutton->setUp	945
SWFDisplayItem	946
SWFDisplayItem->addColor	946
SWFDisplayItem->move	946
SWFDisplayItem->moveTo	947
SWFDisplayItem->multColor	947
SWFDisplayItem->remove	948
SWFDisplayItem->Rotate	949
SWFDisplayItem->rotateTo	949
SWFDisplayItem->scale	951
SWFDisplayItem->scaleTo	952
SWFDisplayItem->setDepth	952
SWFDisplayItem->setName	953
SWFDisplayItem->setRatio	953
SWFDisplayItem->skewX	955
SWFDisplayItem->skewXTo	955
SWFDisplayItem->skewY	956
SWFDisplayItem->skewYTo	956
SWFFill	956
SWFFill->moveTo	957
SWFFill->rotateTo	957
SWFFill->scaleTo	957
SWFFill->skewXTo	958
SWFFill->skewYTo	958
SWFFont	958
swffont->getwidth	959

SWFGradient	960
SWFGradient->addEntry	961
SWFMorph	961
SWFMorph->getshape1	963
SWFMorph->getshape2	963
SWFMovie	964
SWFMovie->add	964
SWFMovie->nextframe	964
SWFMovie->output	965
SWFMovie->remove	965
SWFMovie->save	966
SWFMovie->setbackground	966
SWFMovie->setdimension	966
SWFMovie->setframes	967
SWFMovie->setrate	967
SWFMovie->streammp3	967
SWFShape	968
SWFShape->addFill	969
SWFShape->drawCurve	971
SWFShape->drawCurveTo	972
SWFShape->drawLine	972
SWFShape->drawLineTo	973
SWFShape->movePen	973
SWFShape->movePenTo	974
SWFShape->setLeftFill	974
SWFShape->setLine	975
SWFShape->setRightFill	976
SWFSprite	977
SWFSprite->add	978
SWFSprite->nextframe	978
SWFSprite->remove	979
SWFSprite->setframes	979
SWFText	979
SWFText->addString	980
SWFText->getWidth	981
SWFText->moveTo	981
SWFText->setColor	981
SWFText->setFont	982
SWFText->setHeight	982
SWFText->setSpacing	982
SWFTextField	983
SWFTextField->addstring	984
SWFTextField->align	984
SWFTextField->setbounds	984
SWFTextField->setcolor	985
SWFTextField->setFont	985
SWFTextField->setHeight	985
SWFTextField->setindentation	986

SWFTextField->setLeftMargin	986
SWFTextField->setLineSpacing	986
SWFTextField->setMargins	987
SWFTextField->setname	987
SWFTextField->setrightMargin	987
LIX. Miscellaneous functions	989
connection_aborted	990
connection_status	990
connection_timeout	990
constant	990
define	991
defined	992
die	992
eval	992
exit	993
get_browser	994
highlight_file	995
highlight_string	997
ignore_user_abort	997
iptcpase	998
leak	998
pack	998
show_source	1000
sleep	1000
uniqid	1000
unpack	1001
usleep	1002
LX. mnoGoSearch Functions	1003
udm_add_search_limit	1004
udm_alloc_agent	1004
udm_api_version	1005
udm_cat_list	1006
udm_cat_path	1007
udm_check_charset	1008
udm_check_stored	1008
udm_clear_search_limits	1008
udm_close_stored	1008
udm_crc32	1009
udm_errno	1009
udm_error	1009
udm_find	1009
udm_free_agent	1010
udm_free_ispell_data	1010
udm_free_res	1011
udm_get_doc_count	1011
udm_get_res_field	1011
udm_get_res_param	1012
udm_load_ispell_data	1012

udm_open_stored	1015
udm_set_agent_param	1015
LXI. mSQL functions	1019
msql	1020
msql_affected_rows	1020
msql_close	1020
msql_connect	1020
msql_create_db	1021
msql_createdb	1021
msql_data_seek	1021
msql_dbname	1021
msql_drop_db	1022
msql_dropdb	1022
msql_error	1022
msql_fetch_array	1022
msql_fetch_field	1023
msql_fetch_object	1023
msql_fetch_row	1024
msql_field_seek	1024
msql_fieldflags	1024
msql_fieldlen	1025
msql_fieldname	1025
msql_fieldtable	1025
msql_fieldtype	1025
msql_free_result	1025
msql_freeresult	1026
msql_list_dbs	1026
msql_list_fields	1026
msql_list_tables	1026
msql_listdbs	1027
msql_listfields	1027
msql_listtables	1027
msql_num_fields	1027
msql_num_rows	1027
msql_numfields	1028
msql_numrows	1028
msql_pconnect	1028
msql_query	1028
msql_regcase	1029
msql_result	1029
msql_select_db	1030
msql_selectdb	1030
msql_tablename	1030
LXII. MySQL Functions	1032
mysql_affected_rows	1036
mysql_change_user	1037
mysql_character_set_name	1037
mysql_close	1038

mysql_connect.....	1039
mysql_create_db.....	1040
mysql_data_seek.....	1040
mysql_db_name.....	1042
mysql_db_query.....	1042
mysql_drop_db.....	1043
mysql_errno.....	1043
mysql_error.....	1044
mysql_escape_string.....	1045
mysql_fetch_array.....	1045
mysql_fetch_assoc.....	1047
mysql_fetch_field.....	1048
mysql_fetch_lengths.....	1049
mysql_fetch_object.....	1050
mysql_fetch_row.....	1050
mysql_field_flags.....	1051
mysql_field_len.....	1051
mysql_field_name.....	1051
mysql_field_seek.....	1052
mysql_field_table.....	1052
mysql_field_type.....	1053
mysql_free_result.....	1054
mysql_get_client_info.....	1054
mysql_get_host_info.....	1055
mysql_get_proto_info.....	1055
mysql_get_server_info.....	1056
mysql_info.....	1056
mysql_insert_id.....	1057
mysql_list_dbs.....	1058
mysql_list_fields.....	1059
mysql_list_processes.....	1060
mysql_list_tables.....	1060
mysql_num_fields.....	1061
mysql_num_rows.....	1061
mysql_pconnect.....	1062
mysql_ping.....	1063
mysql_query.....	1063
mysql_real_escape_string.....	1065
mysql_result.....	1065
mysql_select_db.....	1066
mysql_stat.....	1066
mysql_tablename.....	1067
mysql_thread_id.....	1068
mysql_unbuffered_query.....	1068
LXIII. Mohawk Software session handler functions.....	1070
msession_connect.....	1071
msession_count.....	1071
msession_create.....	1071

msession_destroy.....	1071
msession_disconnect.....	1072
msession_find.....	1072
msession_get.....	1072
msession_get_array.....	1072
msession_getdata.....	1073
msession_inc.....	1073
msession_list.....	1073
msession_listvar.....	1073
msession_lock.....	1073
msession_plugin.....	1074
msession_randstr.....	1074
msession_set.....	1074
msession_set_array.....	1075
msession_setdata.....	1075
msession_timeout.....	1075
msession_uniq.....	1075
msession_unlock.....	1076
LXIV. muscat functions.....	1077
muscat_close.....	1078
muscat_get.....	1078
muscat_give.....	1078
muscat_setup.....	1079
muscat_setup_net.....	1079
LXV. Network Functions.....	1081
checkdnsrr.....	1082
closelog.....	1082
debugger_off.....	1082
debugger_on.....	1082
define_syslog_variables.....	1082
fsockopen.....	1083
gethostbyaddr.....	1084
gethostbyname.....	1084
gethostbyname1.....	1085
getmxrr.....	1085
getprotobyname.....	1085
getprotobyname.....	1085
getservbyname.....	1086
getservbyport.....	1086
ip2long.....	1086
long2ip.....	1087
openlog.....	1087
psockopen.....	1089
socket_get_status.....	1089
socket_set_blocking.....	1089
socket_set_timeout.....	1089
syslog.....	1090
LXVI. Nurses terminal screen control functions.....	1092

<code>ncurses_addch</code>	1097
<code>ncurses_addchnstr</code>	1097
<code>ncurses_addchstr</code>	1097
<code>ncurses_addnstr</code>	1097
<code>ncurses_addstr</code>	1098
<code>ncurses_assume_default_colors</code>	1098
<code>ncurses_attroff</code>	1098
<code>ncurses_attron</code>	1099
<code>ncurses_attrset</code>	1099
<code>ncurses_baudrate</code>	1099
<code>ncurses_beep</code>	1100
<code>ncurses_bkgd</code>	1100
<code>ncurses_bkgdset</code>	1100
<code>ncurses_border</code>	1101
<code>ncurses_can_change_color</code>	1101
<code>ncurses_cbreak</code>	1101
<code>ncurses_clear</code>	1102
<code>ncurses_clrtobot</code>	1102
<code>ncurses_clrtoeol</code>	1103
<code>ncurses_color_set</code>	1103
<code>ncurses_curs_set</code>	1103
<code>ncurses_def_prog_mode</code>	1104
<code>ncurses_def_shell_mode</code>	1104
<code>ncurses_define_key</code>	1105
<code>ncurses_delay_output</code>	1105
<code>ncurses_delch</code>	1105
<code>ncurses_deleteln</code>	1106
<code>ncurses_delwin</code>	1106
<code>ncurses_douupdate</code>	1106
<code>ncurses_echo</code>	1107
<code>ncurses_echochar</code>	1107
<code>ncurses_end</code>	1107
<code>ncurses_erase</code>	1108
<code>ncurses_erasechar</code>	1108
<code>ncurses_filter</code>	1109
<code>ncurses_flash</code>	1109
<code>ncurses_flushinp</code>	1109
<code>ncurses_getch</code>	1110
<code>ncurses_getmouse</code>	1110
<code>ncurses_halfdelay</code>	1111
<code>ncurses_has_colors</code>	1111
<code>ncurses_has_ic</code>	1112
<code>ncurses_has_il</code>	1112
<code>ncurses_has_key</code>	1112
<code>ncurses_hline</code>	1113
<code>ncurses_inch</code>	1113
<code>ncurses_init</code>	1113
<code>ncurses_init_color</code>	1114

ncurses_init_pair.....	1114
ncurses_insch.....	1114
ncurses_insdelln.....	1115
ncurses_insertln.....	1115
ncurses_insstr.....	1115
ncurses_instr.....	1116
ncurses_isendwin.....	1116
ncurses_keyok.....	1117
ncurses_killchar.....	1117
ncurses_longname.....	1117
ncurses_mouseinterval.....	1118
ncurses_mousemask.....	1118
ncurses_move.....	1119
ncurses_mvaddch.....	1120
ncurses_mvaddchnstr.....	1120
ncurses_mvaddchstr.....	1120
ncurses_mvaddnstr.....	1121
ncurses_mvaddstr.....	1121
ncurses_mvcur.....	1121
ncurses_mvdelch.....	1122
ncurses_mvgetch.....	1122
ncurses_mvhline.....	1122
ncurses_mvinch.....	1123
ncurses_mvvline.....	1123
ncurses_mvwaddstr.....	1123
ncurses_napms.....	1124
ncurses_newwin.....	1124
ncurses_nl.....	1124
ncurses_nocbreak.....	1125
ncurses_noecho.....	1125
ncurses_nonl.....	1126
ncurses_noqiflush.....	1126
ncurses_noraw.....	1126
ncurses_putp.....	1127
ncurses_qiflush.....	1127
ncurses_raw.....	1127
ncurses_refresh.....	1128
ncurses_resetty.....	1128
ncurses_savetty.....	1129
ncurses_scr_dump.....	1129
ncurses_scr_init.....	1129
ncurses_scr_restore.....	1130
ncurses_scr_set.....	1130
ncurses_scri.....	1130
ncurses_slk_attr.....	1131
ncurses_slk_attroff.....	1131
ncurses_slk_attron.....	1131
ncurses_slk_attrset.....	1132

ncurses_slk_clear.....	1132
ncurses_slk_color.....	1132
ncurses_slk_init.....	1133
ncurses_slk_noutrefresh.....	1133
ncurses_slk_refresh.....	1133
ncurses_slk_restore.....	1134
ncurses_slk_touch.....	1134
ncurses_standend.....	1135
ncurses_standout.....	1135
ncurses_start_color.....	1135
ncurses_termattrs.....	1135
ncurses_termname.....	1136
ncurses_timeout.....	1136
ncurses_typeahead.....	1136
ncurses_ungetch.....	1137
ncurses_ungetmouse.....	1137
ncurses_use_default_colors.....	1138
ncurses_use_env.....	1138
ncurses_use_extended_names.....	1138
ncurses_vidattr.....	1139
ncurses_vline.....	1139
ncurses_wrefresh.....	1139
LXVII. Lotus Notes functions.....	1141
notes_body.....	1142
notes_copy_db.....	1142
notes_create_db.....	1142
notes_create_note.....	1143
notes_drop_db.....	1143
notes_find_note.....	1144
notes_header_info.....	1144
notes_list_msgs.....	1145
notes_mark_read.....	1145
notes_mark_unread.....	1145
notes_nav_create.....	1146
notes_search.....	1146
notes_unread.....	1147
notes_version.....	1147
LXVIII. Unified ODBC functions.....	1149
odbc_autocommit.....	1152
odbc_binmode.....	1152
odbc_close.....	1153
odbc_close_all.....	1153
odbc_columnprivileges.....	1153
odbc_columns.....	1154
odbc_commit.....	1155
odbc_connect.....	1155
odbc_cursor.....	1156
odbc_do.....	1156

odbc_error.....	1156
odbc_errormsg.....	1156
odbc_exec.....	1156
odbc_execute.....	1157
odbc_fetch_array.....	1157
odbc_fetch_into.....	1158
odbc_fetch_object.....	1159
odbc_fetch_row.....	1159
odbc_field_len.....	1160
odbc_field_name.....	1160
odbc_field_num.....	1160
odbc_field_precision.....	1160
odbc_field_scale.....	1161
odbc_field_type.....	1161
odbc_foreignkeys.....	1161
odbc_free_result.....	1162
odbc_gettypeinfo.....	1162
odbc_longreadlen.....	1163
odbc_next_result.....	1163
odbc_num_fields.....	1164
odbc_num_rows.....	1164
odbc_pconnect.....	1164
odbc_prepare.....	1165
odbc_primarykeys.....	1165
odbc_procedurecolumns.....	1165
odbc_procedures.....	1166
odbc_result.....	1167
odbc_result_all.....	1167
odbc_rollback.....	1168
odbc_setoption.....	1168
odbc_specialcolumns.....	1169
odbc_statistics.....	1169
odbc_tableprivileges.....	1170
odbc_tables.....	1171
LXIX. Oracle 8 functions.....	1173
OCIBindByName.....	1176
OCICancel.....	1177
OCICollAppend.....	1177
OCICollAssign.....	1178
OCICollAssignElem.....	1178
OCICollGetElem.....	1178
OCICollMax.....	1178
OCICollSize.....	1179
OCICollTrim.....	1179
OCIColumnIsNULL.....	1179
OCIColumnName.....	1180
OCIColumnPrecision.....	1181
OCIColumnScale.....	1181

OCIColumnSize	1181
OCIColumnType	1182
OCIColumnTypeRaw	1183
OCICommit	1183
OCIDefineByName	1184
OCIError	1184
OCIExecute	1185
OCIFetch	1185
OCIFetchInto	1185
OCIFetchStatement	1185
OCIFreeCollection	1186
OCIFreeCursor	1187
OCIFreeDesc	1187
OCIFreeStatement	1187
OCIInternalDebug	1187
OCILoadLob	1188
OCILogOff	1188
OCILogon	1188
OCINewCollection	1190
OCINewCursor	1190
OCINewDescriptor	1192
OCINLogon	1194
OCINumCols	1196
OCIParse	1197
OCIPLogon	1197
OCIResult	1198
OCIRollback	1198
OCIRowCount	1198
OCISaveLob	1199
OCISaveLobFile	1199
OCIServerVersion	1199
OCISetPrefetch	1200
OCIStatementType	1200
OCIWriteLobToFile	1201
LXX. OpenSSL functions	1202
openssl_csr_export	1205
openssl_csr_export_to_file	1205
openssl_csr_new	1205
openssl_csr_sign	1206
openssl_error_string	1206
openssl_free_key	1207
openssl_get_privatekey	1207
openssl_get_publickey	1208
openssl_open	1208
openssl_pkcs7_decrypt	1209
openssl_pkcs7_encrypt	1210
openssl_pkcs7_sign	1211
openssl_pkcs7_verify	1212

openssl_pkey_export	1213
openssl_pkey_export_to_file	1213
openssl_pkey_new	1214
openssl_private_decrypt	1214
openssl_private_encrypt	1215
openssl_public_decrypt	1215
openssl_public_encrypt	1216
openssl_seal	1216
openssl_sign	1217
openssl_verify	1218
openssl_x509_check_private_key	1219
openssl_x509_checkpurpose	1220
openssl_x509_export	1221
openssl_x509_export_to_file	1221
openssl_x509_free	1221
openssl_x509_parse	1222
openssl_x509_read	1222
LXXI. Oracle functions	1224
Ora_Bind	1225
Ora_Close	1225
Ora_ColumnName	1225
Ora_ColumnSize	1226
Ora_ColumnType	1226
Ora_Commit	1226
Ora_CommitOff	1226
Ora_CommitOn	1227
Ora_Do	1227
Ora_Error	1227
Ora_ErrorCode	1228
Ora_Exec	1228
Ora_Fetch	1228
Ora_Fetch_Into	1229
Ora_GetColumn	1229
Ora_Logoff	1229
Ora_Logon	1230
Ora_Numcols	1230
Ora_Numrows	1230
Ora_Open	1231
Ora_Parse	1231
Ora_pLogon	1231
Ora_Rollback	1231
LXXII. Ovrimos SQL functions	1233
ovrimos_close	1234
ovrimos_commit	1234
ovrimos_connect	1234
ovrimos_cursor	1235
ovrimos_exec	1235
ovrimos_execute	1235

ovrimos_fetch_into.....	1235
ovrimos_fetch_row.....	1236
ovrimos_field_len.....	1237
ovrimos_field_name.....	1238
ovrimos_field_num.....	1238
ovrimos_field_type.....	1238
ovrimos_free_result.....	1238
ovrimos_longreadlen.....	1239
ovrimos_num_fields.....	1239
ovrimos_num_rows.....	1239
ovrimos_prepare.....	1239
ovrimos_result.....	1240
ovrimos_result_all.....	1240
ovrimos_rollback.....	1242
LXXIII. Output Control Functions.....	1243
flush.....	1244
ob_clean.....	1244
ob_end_clean.....	1244
ob_end_flush.....	1244
ob_flush.....	1245
ob_get_contents.....	1245
ob_get_length.....	1245
ob_get_level.....	1245
ob_gzhandler.....	1246
ob_implicit_flush.....	1246
ob_start.....	1247
LXXIV. Object property and method call overloading.....	1249
overload.....	1251
LXXV. PDF functions.....	1252
pdf_add_annotation.....	1258
pdf_add_bookmark.....	1258
pdf_add_launchlink.....	1258
pdf_add_locallink.....	1258
pdf_add_note.....	1258
pdf_add_outline.....	1259
pdf_add_pdflink.....	1259
pdf_add_thumbnail.....	1259
pdf_add_weblink.....	1259
pdf_arc.....	1259
pdf_arcn.....	1260
pdf_attach_file.....	1260
pdf_begin_page.....	1260
pdf_begin_pattern.....	1261
pdf_begin_template.....	1261
pdf_circle.....	1261
pdf_clip.....	1261
pdf_close.....	1262
pdf_close_image.....	1262

pdf_close_pdi	1262
pdf_close_pdi_page	1262
pdf_closepath	1262
pdf_closepath_fill_stroke	1263
pdf_closepath_stroke	1263
pdf_concat	1263
pdf_continue_text	1263
pdf_curveto	1263
pdf_delete	1264
pdf_end_page	1264
pdf_end_pattern	1264
pdf_end_template	1264
pdf_endpath	1264
pdf_fill	1265
pdf_fill_stroke	1265
pdf_findfont	1265
pdf_get_buffer	1266
pdf_get_font	1266
pdf_get_fontname	1266
pdf_get_fontsize	1266
pdf_get_image_height	1266
pdf_get_image_width	1266
pdf_get_majorversion	1267
pdf_get_minorversion	1267
pdf_get_parameter	1267
pdf_get_pdi_parameter	1267
pdf_get_pdi_value	1267
pdf_get_value	1268
pdf_initgraphics	1268
pdf_lineto	1268
pdf_makespotcolor	1268
pdf_moveto	1268
pdf_new	1269
pdf_open	1269
pdf_open_CCITT	1269
pdf_open_file	1269
pdf_open_gif	1270
pdf_open_image	1270
pdf_open_image_file	1271
pdf_open_jpeg	1271
pdf_open_memory_image	1271
pdf_open_pdi	1272
pdf_open_pdi_page	1272
pdf_open_png	1272
pdf_open_tiff	1272
pdf_place_image	1273
pdf_place_pdi_page	1273
pdf_rect	1273

pdf_restore.....	1273
pdf_rotate.....	1273
pdf_save.....	1274
pdf_scale.....	1274
pdf_set_border_color.....	1274
pdf_set_border_dash.....	1274
pdf_set_border_style.....	1274
pdf_set_char_spacing.....	1275
pdf_set_duration.....	1275
pdf_set_font.....	1275
pdf_set_horiz_scaling.....	1275
pdf_set_info.....	1275
pdf_set_info_author.....	1276
pdf_set_info_creator.....	1276
pdf_set_info_keywords.....	1276
pdf_set_info_subject.....	1276
pdf_set_info_title.....	1276
pdf_set_leading.....	1277
pdf_set_parameter.....	1277
pdf_set_text_pos.....	1277
pdf_set_text_rendering.....	1277
pdf_set_text_rise.....	1277
pdf_set_text_matrix.....	1278
pdf_set_value.....	1278
pdf_set_word_spacing.....	1278
pdf_setcolor.....	1278
pdf_setdash.....	1279
pdf_setflat.....	1279
pdf_setfont.....	1279
pdf_setgray.....	1280
pdf_setgray_fill.....	1280
pdf_setgray_stroke.....	1280
pdf_setlinecap.....	1281
pdf_setlinejoin.....	1281
pdf_setlinewidth.....	1281
pdf_setmatrix.....	1281
pdf_setmiterlimit.....	1281
pdf_setpolydash.....	1281
pdf_setrgbcolor.....	1282
pdf_setrgbcolor_fill.....	1282
pdf_setrgbcolor_stroke.....	1282
pdf_show.....	1282
pdf_show_boxed.....	1283
pdf_show_xy.....	1283
pdf_skew.....	1283
pdf_stringwidth.....	1283
pdf_stroke.....	1284
pdf_translate.....	1284

LXXVI. Verisign Payflow Pro functions.....	1285
pfpro_cleanup.....	1286
pfpro_init.....	1286
pfpro_process.....	1286
pfpro_process_raw.....	1287
pfpro_version.....	1288
LXXVII. PHP Options&Information.....	1289
assert.....	1290
assert_options.....	1291
dl.....	1291
extension_loaded.....	1291
get_cfg_var.....	1292
get_current_user.....	1292
get_defined_constants.....	1292
get_extension_funcs.....	1293
get_included_files.....	1294
get_loaded_extensions.....	1295
get_magic_quotes_gpc.....	1296
get_magic_quotes_runtime.....	1296
get_required_files.....	1296
getenv.....	1296
getlastmod.....	1297
getmygid.....	1297
getmyinode.....	1297
getmypid.....	1298
getmyuid.....	1298
getrusage.....	1298
ini_alter.....	1299
ini_get.....	1299
ini_get_all.....	1299
ini_restore.....	1300
ini_set.....	1300
php_logo_guid.....	1307
php_sapi_name.....	1307
php_uname.....	1308
phpcredits.....	1308
phpinfo.....	1310
phpversion.....	1311
putenv.....	1312
set_magic_quotes_runtime.....	1313
set_time_limit.....	1313
version_compare.....	1313
zend_logo_guid.....	1314
zend_version.....	1314
LXXVIII. POSIX functions.....	1316
posix_ctermid.....	1317
posix_getcwd.....	1317
posix_getegid.....	1317

posix_geteuid.....	1317
posix_getgid	1317
posix_getgrgid.....	1318
posix_getgrnam	1318
posix_getgroups.....	1318
posix_getlogin	1318
posix_getpgid	1318
posix_getpgrp	1319
posix_getpid	1319
posix_getppid	1319
posix_getpwnam.....	1319
posix_getpwuid.....	1320
posix_getrlimit.....	1321
posix_getsid.....	1321
posix_getuid	1322
posix_isatty.....	1322
posix_kill	1322
posix_mkfifo.....	1322
posix_setegid	1323
posix_seteuid	1323
posix_setgid.....	1323
posix_setpgid.....	1323
posix_setsid	1324
posix_setuid	1324
posix_times.....	1324
posix_ttyname.....	1325
posix_uname.....	1325
LXXIX. PostgreSQL functions.....	1326
pg_affected_rows.....	1330
pg_cancel_query.....	1330
pg_client_encoding.....	1330
pg_close.....	1331
pg_connect.....	1331
pg_connection_busy	1332
pg_connection_reset	1332
pg_connection_status	1332
pg_convert	1333
pg_copy_from.....	1333
pg_copy_to	1333
pg_dbname	1333
pg_delete.....	1334
pg_end_copy.....	1334
pg_escape_bytea.....	1335
pg_escape_string	1335
pg_fetch_array.....	1335
pg_fetch_object	1337
pg_fetch_result	1338
pg_fetch_row	1339

pg_field_is_null	1340
pg_field_name	1340
pg_field_num	1340
pg_field_prtlen	1341
pg_field_size	1341
pg_field_type	1341
pg_free_result	1342
pg_get_result	1342
pg_host	1342
pg_insert	1343
pg_last_error	1343
pg_last_notice	1344
pg_last_oid	1344
pg_lo_close	1345
pg_lo_create	1345
pg_lo_export	1345
pg_lo_import	1346
pg_lo_open	1346
pg_lo_read	1347
pg_lo_read_all	1347
pg_lo_seek	1347
pg_lo_tell	1348
pg_lo_unlink	1348
pg_lo_write	1348
pg_metadata	1349
pg_num_fields	1349
pg_num_rows	1349
pg_options	1350
pg_pconnect	1350
pg_port	1350
pg_put_line	1350
pg_query	1351
pg_result_error	1352
pg_result_status	1352
pg_select	1352
pg_send_query	1353
pg_set_client_encoding	1353
pg_trace	1354
pg_tty	1354
pg_untrace	1355
pg_update	1355
LXXX. Process Control Functions	1356
pcntl_exec	1358
pcntl_fork	1358
pcntl_signal	1358
pcntl_waitpid	1360
pcntl_wexitstatus	1361
pcntl_wifexited	1361

pcntl_wifsignaled	1361
pcntl_wifstopped	1361
pcntl_wstpsig	1362
pcntl_wtermsig	1362
LXXXI. Program Execution functions	1363
escapshellarg	1364
escapshellcmd	1364
exec	1364
passthru	1365
proc_close	1366
proc_open	1366
shell_exec	1367
system	1367
LXXXII. Printer functions	1369
printer_abort	1370
printer_close	1370
printer_create_brush	1370
printer_create_dc	1371
printer_create_font	1372
printer_create_pen	1372
printer_delete_brush	1373
printer_delete_dc	1373
printer_delete_font	1373
printer_delete_pen	1373
printer_draw_bmp	1373
printer_draw_chord	1374
printer_draw_ellipse	1375
printer_draw_line	1376
printer_draw_pie	1376
printer_draw_rectangle	1377
printer_draw_roundrect	1378
printer_draw_text	1379
printer_end_doc	1380
printer_end_page	1380
printer_get_option	1380
printer_list	1380
printer_logical_fontheight	1381
printer_open	1382
printer_select_brush	1382
printer_select_font	1383
printer_select_pen	1383
printer_set_option	1384
printer_start_doc	1386
printer_start_page	1386
printer_write	1386
LXXXIII. Pspell Functions	1388
pspell_add_to_personal	1389
pspell_add_to_session	1389

pspell_check	1389
pspell_clear_session	1390
pspell_config_create	1390
pspell_config_ignore	1391
pspell_config_mode.....	1392
pspell_config_personal	1392
pspell_config_repl	1393
pspell_config_runtogether	1393
pspell_config_save_repl	1394
pspell_new.....	1394
pspell_new_config.....	1395
pspell_new_personal	1396
pspell_save_wordlist.....	1397
pspell_store_replacement	1397
pspell_suggest.....	1398
LXXXIV. GNU Readline	1400
readline	1401
readline_add_history	1401
readline_clear_history	1401
readline_completion_function.....	1401
readline_info.....	1402
readline_list_history	1402
readline_read_history	1402
readline_write_history	1402
LXXXV. GNU Recode functions	1404
recode	1405
recode_file	1405
recode_string	1405
LXXXVI. Regular Expression Functions (Perl-Compatible)	1407
Pattern Modifiers	1409
Pattern Syntax.....	1410
preg_grep.....	1436
preg_match	1437
preg_match_all	1438
preg_quote	1441
preg_replace.....	1441
preg_replace_callback	1444
preg_split	1444
LXXXVII. qtdom functions	1446
qdom_error	1447
qdom_tree	1447
LXXXVIII. Regular Expression Functions (POSIX Extended)	1448
ereg	1450
ereg_replace.....	1450
eregi	1452
eregi_replace.....	1452
split	1453
spliti	1454

sql_regcase.....	1454
LXXXIX. Semaphore and Shared Memory Functions.....	1455
ftok.....	1456
sem_acquire.....	1456
sem_get.....	1456
sem_release.....	1457
sem_remove.....	1457
shm_attach.....	1457
shm_detach.....	1458
shm_get_var.....	1458
shm_put_var.....	1458
shm_remove.....	1459
shm_remove_var.....	1459
XC. SESAM database functions.....	1460
sesam_affected_rows.....	1466
sesam_commit.....	1466
sesam_connect.....	1467
sesam_diagnostic.....	1467
sesam_disconnect.....	1469
sesam_errormsg.....	1470
sesam_execimm.....	1470
sesam_fetch_array.....	1471
sesam_fetch_result.....	1473
sesam_fetch_row.....	1473
sesam_field_array.....	1475
sesam_field_name.....	1478
sesam_free_result.....	1478
sesam_num_fields.....	1478
sesam_query.....	1479
sesam_rollback.....	1480
sesam_seek_row.....	1481
sesam_settransaction.....	1482
XCI. Session handling functions.....	1484
session_cache_expire.....	1490
session_cache_limiter.....	1490
session_decode.....	1491
session_destroy.....	1491
session_encode.....	1492
session_get_cookie_params.....	1492
session_id.....	1492
session_is_registered.....	1493
session_module_name.....	1493
session_name.....	1493
session_readonly.....	1494
session_register.....	1494
session_save_path.....	1495
session_set_cookie_params.....	1495
session_set_save_handler.....	1496

session_start.....	1498
session_unregister.....	1498
session_unset.....	1499
session_write_close.....	1499
XCII. Shared Memory Functions.....	1500
shmop_close.....	1501
shmop_delete.....	1501
shmop_open.....	1501
shmop_read.....	1502
shmop_size.....	1503
shmop_write.....	1503
XCIII. Shockwave Flash functions.....	1505
swf_actiongeturl.....	1507
swf_actiongotoframe.....	1507
swf_actiongotolabel.....	1507
swf_actionnextframe.....	1507
swf_actionplay.....	1507
swf_actionprevframe.....	1508
swf_actionsettarget.....	1508
swf_actionstop.....	1508
swf_actiontogglequality.....	1508
swf_actionwaitforframe.....	1508
swf_addbuttonrecord.....	1508
swf_addcolor.....	1509
swf_closefile.....	1509
swf_definebitmap.....	1511
swf_definefont.....	1511
swf_defineline.....	1511
swf_definepoly.....	1512
swf_definerect.....	1512
swf_definetext.....	1512
swf_endbutton.....	1512
swf_enddoaction.....	1513
swf_endshape.....	1513
swf_endsymbol.....	1513
swf_fontsize.....	1513
swf_fontslant.....	1513
swf_fontracking.....	1514
swf_getbitmapinfo.....	1514
swf_getfontinfo.....	1514
swf_getframe.....	1515
swf_labelframe.....	1515
swf_lookat.....	1515
swf_modifyobject.....	1515
swf_mulcolor.....	1516
swf_nextid.....	1516
swf_oncondition.....	1516
swf_openfile.....	1517

swf_ortho.....	1517
swf_ortho2.....	1517
swf_perspective.....	1518
swf_placeobject.....	1518
swf_polarview.....	1518
swf_popmatrix.....	1519
swf_posround.....	1519
swf_pushmatrix.....	1519
swf_removeobject.....	1519
swf_rotate.....	1520
swf_scale.....	1520
swf_setfont.....	1520
swf_setframe.....	1520
swf_shapearc.....	1521
swf_shapecurveto.....	1521
swf_shapecurveto3.....	1521
swf_shapefillbitmapclip.....	1521
swf_shapefillbitmaptile.....	1521
swf_shapefilloff.....	1522
swf_shapefillsolid.....	1522
swf_shapelinesolid.....	1522
swf_shapelineto.....	1522
swf_shapemoveto.....	1523
swf_showframe.....	1523
swf_startbutton.....	1523
swf_startdoaction.....	1523
swf_startshape.....	1523
swf_startsymbol.....	1524
swf_textwidth.....	1524
swf_translate.....	1524
swf_viewport.....	1524
XCIV. SNMP functions.....	1525
snmp_get_quick_print.....	1526
snmp_set_quick_print.....	1526
snmpget.....	1527
snmprealwalk.....	1527
snmpset.....	1527
snmpwalk.....	1528
snmpwalkoid.....	1528
XCV. Socket functions.....	1530
socket_accept.....	1533
socket_bind.....	1533
socket_close.....	1534
socket_connect.....	1534
socket_create.....	1535
socket_create_listen.....	1535
socket_create_pair.....	1536
socket_fd_alloc.....	1536

socket_fd_clear.....	1537
socket_fd_free.....	1537
socket_fd_isset.....	1537
socket_fd_set.....	1538
socket_fd_zero.....	1538
socket_getopt.....	1539
socket_getpeername.....	1539
socket_getsockname.....	1540
socket_iovec_add.....	1540
socket_iovec_alloc.....	1541
socket_iovec_delete.....	1541
socket_iovec_fetch.....	1542
socket_iovec_free.....	1542
socket_iovec_set.....	1543
socket_last_error.....	1543
socket_listen.....	1543
socket_read.....	1544
socket_readv.....	1544
socket_recv.....	1545
socket_recvfrom.....	1545
socket_recvmsg.....	1546
socket_select.....	1546
socket_send.....	1547
socket_sendmsg.....	1547
socket_sendto.....	1548
socket_set_nonblock.....	1548
socket_setopt.....	1549
socket_shutdown.....	1549
socket_strerror.....	1549
socket_write.....	1550
socket_writev.....	1551
XCVI. String functions.....	1552
addslashes.....	1553
addslashes.....	1553
bin2hex.....	1554
chop.....	1554
chr.....	1554
chunk_split.....	1555
convert_cyr_string.....	1555
count_chars.....	1556
crc32.....	1556
crypt.....	1556
echo.....	1558
explode.....	1559
get_html_translation_table.....	1560
get_meta_tags.....	1561
hebrew.....	1561
hebrevc.....	1561

htmlentities	1562
htmlspecialchars	1562
implode	1563
join	1564
levenshtein	1564
localeconv	1565
ltrim	1567
md5	1568
md5_file	1568
metaphone	1569
nl_langinfo	1569
nl2br	1569
ord	1570
parse_str	1570
print	1571
printf	1572
quoted_printable_decode	1572
quotemeta	1572
rtrim	1573
setlocale	1574
similar_text	1574
soundex	1575
sprintf	1575
sscanf	1578
str_pad	1579
str_repeat	1579
str_replace	1580
str_rot13	1580
strcasecmp	1581
strchr	1581
strcmp	1581
strcoll	1581
strcspn	1582
strip_tags	1582
stripslashes	1583
stripslashes	1583
stristr	1583
strlen	1584
strnatcasecmp	1584
strnatcmp	1584
strncasecmp	1585
strncmp	1585
strpos	1586
strrchr	1586
strrev	1587
strrpos	1587
strspn	1588
strstr	1589

strtok	1589
strtolower	1591
strtoupper	1591
strtr	1591
substr	1592
substr_count	1593
substr_replace	1594
trim	1595
ucfirst	1596
ucwords	1596
vprintf	1597
vsprintf	1597
wordwrap	1597
XCVII. Sybase functions	1600
sybase_affected_rows	1601
sybase_close	1601
sybase_connect	1601
sybase_data_seek	1602
sybase_fetch_array	1602
sybase_fetch_field	1602
sybase_fetch_object	1603
sybase_fetch_row	1603
sybase_field_seek	1604
sybase_free_result	1604
sybase_get_last_message	1604
sybase_min_client_severity	1604
sybase_min_error_severity	1605
sybase_min_message_severity	1605
sybase_min_server_severity	1605
sybase_num_fields	1605
sybase_num_rows	1605
sybase_pconnect	1606
sybase_query	1606
sybase_result	1606
sybase_select_db	1607
XCVIII. URL Functions	1608
base64_decode	1609
base64_encode	1609
parse_url	1609
rawurldecode	1609
rawurlencode	1610
urldecode	1611
urlencode	1611
XCIX. Variable Functions	1613
doubleval	1614
empty	1614
floatval	1614
get_defined_vars	1615

get_resource_type.....	1616
gettype	1616
import_request_variables.....	1617
intval	1618
is_array	1618
is_bool	1618
is_callable.....	1618
is_double.....	1619
is_float	1619
is_int	1619
is_integer	1620
is_long	1620
is_null	1620
is_numeric	1620
is_object.....	1620
is_real	1621
is_resource.....	1621
is_scalar	1621
is_string	1622
isset.....	1622
print_r.....	1623
serialize.....	1624
settype.....	1625
strval	1626
unserialize.....	1626
unset.....	1628
var_dump.....	1630
var_export.....	1631
C. vpopmail functions	1633
vpopmail_add_alias_domain.....	1634
vpopmail_add_alias_domain_ex	1634
vpopmail_add_domain	1634
vpopmail_add_domain_ex	1635
vpopmail_add_user.....	1635
vpopmail_alias_add.....	1636
vpopmail_alias_del.....	1636
vpopmail_alias_del_domain.....	1637
vpopmail_alias_get.....	1637
vpopmail_alias_get_all.....	1638
vpopmail_auth_user	1638
vpopmail_del_domain	1638
vpopmail_del_domain_ex	1639
vpopmail_del_user	1639
vpopmail_error	1640
vpopmail_passwd	1640
vpopmail_set_user_quota.....	1641
CI. W32api functions	1642
w32api_deftype	1643

w32api_init_dtype	1643
w32api_invoke_function	1643
w32api_register_function	1644
w32api_set_call_method	1644
CII. WDDX Functions	1646
wddx_add_vars	1648
wddx_deserialize	1648
wddx_packet_end	1648
wddx_packet_start	1648
wddx_serialize_value	1648
wddx_serialize_vars	1649
CIII. XML parser functions	1650
utf8_decode	1659
utf8_encode	1659
xml_error_string	1659
xml_get_current_byte_index	1660
xml_get_current_column_number	1660
xml_get_current_line_number	1660
xml_get_error_code	1660
xml_parse	1661
xml_parse_into_struct	1661
xml_parser_create	1665
xml_parser_create_ns	1666
xml_parser_free	1666
xml_parser_get_option	1666
xml_parser_set_option	1667
xml_set_character_data_handler	1667
xml_set_default_handler	1668
xml_set_element_handler	1668
xml_set_end_namespace_decl_handler	1670
xml_set_external_entity_ref_handler	1670
xml_set_notation_decl_handler	1671
xml_set_object	1672
xml_set_processing_instruction_handler	1673
xml_set_start_namespace_decl_handler	1674
xml_set_unparsed_entity_decl_handler	1674
CIV. XMLRPC functions	1676
xmlrpc_decode	1677
xmlrpc_decode_request	1677
xmlrpc_encode	1677
xmlrpc_encode_request	1678
xmlrpc_get_type	1678
xmlrpc_parse_method_descriptions	1679
xmlrpc_server_add_introspection_data	1679
xmlrpc_server_call_method	1680
xmlrpc_server_create	1680
xmlrpc_server_destroy	1680
xmlrpc_server_register_introspection_callback	1681

xmlrpc_server_register_method	1681
xmlrpc_set_type	1682
CV. XSLT functions.....	1683
xslt_create.....	1684
xslt_errno.....	1684
xslt_error.....	1684
xslt_free	1684
xslt_process	1685
xslt_set_base.....	1687
xslt_set_encoding	1687
xslt_set_error_handler	1688
xslt_set_log.....	1688
xslt_set_sax_handler.....	1689
xslt_set_sax_handlers	1690
xslt_set_scheme_handler.....	1690
xslt_set_scheme_handlers	1690
CVI. YAZ functions	1692
yaz_addinfo	1694
yaz_ccl_conf.....	1694
yaz_ccl_parse	1694
yaz_close	1695
yaz_connect	1695
yaz_database.....	1696
yaz_element.....	1696
yaz_errno	1696
yaz_error.....	1697
yaz_hits.....	1697
yaz_itemorder.....	1697
yaz_present.....	1699
yaz_range.....	1700
yaz_record	1700
yaz_scan	1700
yaz_scan_result.....	1701
yaz_search	1702
yaz_sort.....	1703
yaz_syntax	1704
yaz_wait.....	1704
CVII. YP/NIS Functions	1705
yp_all	1706
yp_cat	1706
yp_err_string.....	1706
yp_errno.....	1707
yp_first.....	1707
yp_get_default_domain	1708
yp_master	1708
yp_match	1709
yp_next	1709
yp_order.....	1710

CVIII. Zip File Functions (Read Only Access)	1711
zip_close	1713
zip_entry_close	1713
zip_entry_compressedsize	1713
zip_entry_compressionmethod	1713
zip_entry_filesize	1713
zip_entry_name	1714
zip_entry_open	1714
zip_entry_read	1714
zip_open	1715
zip_read	1715
CIX. Zlib Compression Functions	1716
gzclose	1718
gzcompress	1718
gzdeflate	1718
gzencode	1718
gzeof	1719
gzfile	1720
gzgetc	1720
gzgets	1720
gzgetss	1720
gzinflate	1721
gzopen	1721
gzpassthru	1722
gzputs	1722
gzread	1722
gzrewind	1723
gzseek	1723
gztell	1723
gzuncompress	1724
gzwrite	1724
readgzfile	1724
V. Extending PHP 4.0	1726
24. Overview	1726
What Is Zend? and What Is PHP?	1727
25. Extension Possibilities	1728
External Modules	1729
Built-in Modules	1729
The Zend Engine	1730
26. Source Layout	1731
Extension Conventions	1733
Macros	1733
Memory Management	1733
Directory and File Functions	1734
String Handling	1734
Complex Types	1734
27. PHP's Automatic Build System	1736

28. Creating Extensions	1739
Compiling Modules	1741
29. Using Extensions.....	1743
30. Troubleshooting	1746
31. Source Discussion	1748
Module Structure	1749
Header File Inclusions	1749
Declaring Exported Functions	1749
Declaration of the Zend Function Block	1750
Declaration of the Zend Module Block	1752
Creation of get_module()	1753
Implementation of All Exported Functions	1754
Summary.....	1754
32. Accepting Arguments.....	1755
Determining the Number of Arguments	1756
Retrieving Arguments.....	1757
Old way of retrieving arguments (deprecated)	1760
Dealing with a Variable Number of Arguments/Optional Parameters	1761
Accessing Arguments	1763
Dealing with Arguments Passed by Reference.....	1766
Assuring Write Safety for Other Parameters	1767
33. Creating Variables	1769
Overview	1770
Longs (Integers).....	1772
Doubles (Floats)	1773
Strings.....	1773
Booleans	1774
Arrays	1775
Objects.....	1778
Resources.....	1778
Macros for Automatic Global Variable Creation.....	1782
Creating Constants.....	1783
34. Duplicating Variable Contents: The Copy Constructor	1785
35. Returning Values	1787
36. Printing Information.....	1790
zend_printf()	1791
zend_error()	1791
Including Output in <code>phpinfo()</code>	1791
Execution Information.....	1792
37. Startup and Shutdown Functions	1794
38. Calling User Functions.....	1796
39. Initialization File Support	1799
40. Where to Go from Here	1802
41. Reference: Some Configuration Macros	1804
<code>config.m4</code>	1805
42. API Macros	1806

VI. FAQ: Frequently Asked Questions	1808
43. General Information	1808
44. Mailing lists.....	1811
45. Obtaining PHP	1814
46. Database issues	??
47. Installation.....	??
48. Build Problems.....	??
49. Using PHP.....	??
50. PHP and HTML	??
51. PHP and COM	??
52. PHP and other languages	??
53. Migrating from PHP 2 to PHP 3	??
54. Migrating from PHP 3 to PHP 4	??
55. Miscellaneous Questions.....	??
VII. Appendixes	??
A. History of PHP and related projects.....	??
History of PHP	??
PHP/FI	??
PHP 3	??
PHP 4	??
History of PHP related projects.....	??
PEAR	??
PHP Quality Assurance Initiative	??
PHP-GTK.....	??
Books about PHP.....	??
Publications about PHP	??
B. Using PHP from the command line.....	??
C. Migrating from PHP 3 to PHP 4	??
What has changed in PHP 4	??
Running PHP 3 and PHP 4 concurrently.....	??
Migrating Configuration Files	??
Parser behavior	??
Error reporting	??
Configuration changes	??
Additional warning messages	??
Initializers	??
empty("0").....	??
Missing functions	??
Functions missing due to conceptual changes	??
Deprecate functions and extensions.....	??
Changed status for unset()	??
PHP 3 extension	??
Variable substitution in strings	??
Cookies.....	??
Handling of global variables.....	??
D. Migrating from PHP/FI 2 to PHP 3.....	??
About the incompatibilities in 3.0.....	??

Start/end tags	??
if..endif syntax	??
while syntax	??
Expression types	??
Error messages have changed	??
Short-circuited boolean evaluation	??
Function TRUE/FALSE return values	??
Other incompatibilities	??
E. Debugging PHP	??
About the debugger	??
Using the Debugger	??
Debugger Protocol	??
F. Extending PHP 3	??
Adding functions to PHP 3	??
Function Prototype	??
Function Arguments	??
Variable Function Arguments	??
Using the Function Arguments	??
Memory Management in Functions	??
Setting Variables in the Symbol Table	??
Returning simple values	??
Returning complex values	??
Using the resource list	??
Using the persistent resource table	??
Adding runtime configuration directives	??
Calling User Functions	??
HashTable *function_table	??
pval *object	??
pval *function_name	??
pval *retval	??
int param_count	??
pval *params[]	??
Reporting Errors	??
E_NOTICE	??
E_WARNING	??
E_ERROR	??
E_PARSE	??
E_CORE_ERROR	??
E_CORE_WARNING	??
E_COMPILE_ERROR	??
E_COMPILE_WARNING	??
E_USER_ERROR	??
E_USER_WARNING	??
E_USER_NOTICE	??
E_ALL	??
G. List of Function Aliases	??
H. List of Reserved Words	??
List of Keywords	??

Predefined Variables	??
Server variables: \$_SERVER	??
Environment variables: \$_ENV.....	??
HTTP Cookies: \$_COOKIE.....	??
HTTP GET variables: \$_GET	??
HTTP POST variables: \$_POST	??
HTTP File upload variables: \$_FILES.....	??
Request variables: \$_REQUEST.....	??
Session variables: \$_SESSION.....	??
Global variables: \$GLOBALS.....	??
The previous error message: \$php_errormsg.....	??
Predefined Classes	??
Standard Defined Classes.....	??
Ming Defined Classes	??
Oracle 8 Defined Classes	??
qtdom Defined Classes.....	??
Predefined Constants	??
Core Predefined Constants.....	??
dbx Predefined Constants.....	??
domxml Predefined Constants	??
fbsql Predefined Constants.....	??
fdf Predefined Constants	??
fribidi Predefined Constants.....	??
ftp Predefined Constants	??
gd Predefined Constants.....	??
gmp Predefined Constants.....	??
hyperwave Predefined Constants	??
imap Predefined Constants.....	??
ingres Predefined Constants.....	??
interbase Predefined Constants	??
ldap Predefined Constants.....	??
mbstring Predefined Constants	??
mcal Predefined Constants	??
mcrypt Predefined Constants	??
ming Predefined Constants	??
mnogosearch Predefined Constants	??
mysql Predefined Constants.....	??
mssql Predefined Constants	??
ncurses Predefined Constants.....	??
oci8 Predefined Constants.....	??
odbc Predefined Constants.....	??
openssl Predefined Constants.....	??
oracle Predefined Constants.....	??
pcntl Predefined Constants.....	??
pcre Predefined Constants.....	??
pgsql Predefined Constants.....	??
pspell Predefined Constants	??
session Predefined Constants	??

sockets Predefined Constants	??
standard Predefined Constants	??
swf Predefined Constants	??
tokenizer Predefined Constants	??
w32api Predefined Constants	??
xml Predefined Constants	??
yp Predefined Constants	??
zlib Predefined Constants	??
I. List of Resource Types	??
J. About the manual	??
Formats	??
About user notes	??
How to find more information about PHP	??
How to help improve the documentation	??
How we generate the formats	??
K. missing stuff	??

Preface

PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated webpages quickly, but you can do much more with PHP.

This manual consists primarily of a function reference, but also contains a language reference, explanations of some of PHP's major features, and other supplemental information.

You can download this manual in several formats at <http://www.php.net/docs.php>. The downloads are updated as the content changes. More information about how this manual is developed can be found in the 'About the manual' appendix.

Part I. Getting Started

Chapter 1. Introduction

What is PHP?

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

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 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.

The best things in using PHP are that it is extremely simple for a newcomer, but offers many advanced features for a professional programmer. Don't be afraid reading the long list of PHP's features. You can jump in, in a short time, and start writing simple scripts in a few hours.

Although PHP's development is focused on server-side scripting, you can do much more with it. Read on, and see more in the [What can PHP do?](#) section.

What can PHP do?

Anything. PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

There are three main fields where PHP scripts are used.

- Server-side scripting. This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (CGI or server module), a webserver and a web browser. You need to run the webserver, with a connected PHP installation. You can access the PHP program output with a web browser, viewing the PHP page through the server. See the installation instructions section for more information.
- Command line scripting. You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks. See the section about Command line usage of PHP for more information.
- Writing client-side GUI applications. PHP is probably not the very best language to write windowing applications, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution. If you are interested in PHP-GTK, visit it's own website (<http://gtk.php.net/>).

PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS, and probably others. PHP has also support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, O'Reilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others. For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

So with PHP, you have the freedom of choosing an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object oriented programming, or a mixture of them. Although not every standard OOP feature is realized in the current version of PHP, many code libraries and large applications (including the PEAR library) are written only using OOP code.

With PHP you are not limited to output HTML. PHP's abilities includes outputting images, PDF files and even Flash movies (using libswf and Ming) generated on the fly. You can also output easily any text, such as XHTML and any other XML file. PHP can autogenerate these files, and save them in the file system, instead of printing it out, forming a server-side cache for your dynamic content.

One of 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

We also have a DBX database abstraction extension allowing you to transparently use any database supported by that extension. Additionally PHP supports ODBC, the Open Database Connection standard, so you can connect to any other database supporting this world standard.

PHP also has support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (on Windows) and countless others. You can also open raw network sockets and

interact using any other protocol. PHP has support for the WDDX complex data exchange between virtually all Web programming languages. Talking about interconnection, PHP has support for instantiation of Java objects and using them transparently as PHP objects. You can also use our CORBA extension to access remote objects.

PHP has extremely useful text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents. For parsing and accessing XML documents, we support the SAX and DOM standards. You can use our XSLT extension to transform XML documents.

While using PHP in the ecommerce field, you'll find the Cybercash payment, CyberMUT, VeriSign Payflow Pro and CCVS functions useful for your online payment programs.

At last but not least, we have many other interesting extensions, the mnoGoSearch search engine functions, the IRC Gateway functions, many compression utilities (gzip, bz2), calendar conversion, translation...

As you can see this page is not enough to list all the features and benefits PHP can offer. Read on in the sections about installing PHP, and see the function reference part for explanation of the extensions mentioned here.

Chapter 2. Installation

General Installation Considerations

Before installing first, you need to know what do you want to use PHP for. There are three main fields you can use PHP, as described in the What can PHP do? section:

- Server-side scripting
- Command line scripting
- Client-side GUI applications

For the first and most common form, you need three things: PHP itself, a web server and a web browser. You probably already have a web browser, and depending on your operating system setup, you may also have a web server (eg. Apache on Linux or IIS on Windows). You may also rent webspace at a company. This way, you don't need to set up anything on your own, only write your PHP scripts, upload it to the server you rent, and see the results in your browser. You can find a list of hosting companies at <http://hosts.php.net/>.

While setting up the server and PHP on your own, you have two choices for the method of connecting PHP to the server. For many servers PHP has a direct module interface (also called SAPI). These servers include Apache, Microsoft Internet Information Server, Netscape and iPlanet servers. Many other servers have support for ISAPI, the Microsoft module interface (OmniHTTPd for example). If PHP has no module support for your web server, you can always use it as a CGI processor. This means you set up your server to use the command line executable of PHP (`php.exe` on Windows) to process all PHP file requests on the server.

If you are also interested to use PHP for command line scripting (eg. write scripts autogenerating some images for you offline, or processing text files depending on some arguments you pass to them), you always need the command line executable. For more information, read the section about writing command line PHP applications. In this case, you need no server and no browser.

With PHP you can also write client side GUI applications using the PHP-GTK extension. This is a completely different approach than writing web pages, as you do not output any HTML, but manage windows and objects within them. For more information about PHP-GTK, please visit the site dedicated to this extension (<http://gtk.php.net/>). PHP-GTK is not included in the official PHP distribution.

From now on, this section deals with setting up PHP for web servers on Unix and Windows with server module interfaces and CGI executables.

Downloading PHP, the source code, and binary distributions for Windows can be found at <http://www.php.net/>. We recommend you to choose a mirror (<http://www.php.net/mirrors.php>) nearest to you for downloading the distributions.

Unix/HP-UX installs

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

Example 2-1. 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://hpux.connect.org.uk/ftp/hpux/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/hpux/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://hpux.connect.org.uk/ftp/hpux/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://hpux.connect.org.uk/ftp/hpux/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 <filename>non-gnu</filename> 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 hpux machine, there will be a better way of doing this but a cheap and cheerful hack is to put
 lt_target=hpux10.20
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}-hpux\${HPUXVER}"; exit 0"
to:
 "echo "hp\${HPUXMACH}-hp-hpux\${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.

Unix/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 system, 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.

Unix/Mac OS X installs

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

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.

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 2-2. 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 \
--iconsdir=/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 additions (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 you 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/>).

Compiling for MacOS X client

Those tips are graciously provided by Marc Liyanage (<http://www.entropy.ch/software/macosx>).

The PHP module for the Apache web server included in Mac OS X. This version includes support for the MySQL and PostgreSQL databases.

NOTE: Be careful when you do this, you could screw up your Apache web server!

Do this to install:

- 1. Open a terminal window
- 2. Type "`wget http://www.diix.ch/users/liyanage/software/macosx/libphp4.so.gz`", wait for download to finish
- 3. Type "`gunzip libphp4.so.gz`"
- 4. Type "`sudo apxs -i -a -n php4 libphp4.so`"

Now type "`sudo open -a TextEdit /etc/httpd/httpd.conf`" TextEdit will open with the web server configuration file. Locate these two lines towards the end of the file: (Use the Find command)

```
#AddType application/x-httpd-php .php
#AddType application/x-httpd-php-source .phps
```

Remove the two hash marks (#), then save the file and quit TextEdit.

Finally, type "sudo apachectl graceful" to restart the web server.

PHP should now be up and running. You can test it by dropping a file into your "Sites" folder which is called "test.php". Into that file, write this line: "<?php phpinfo() ?>".

Now open up 127.0.0.1/~your_username/test.php in your web browser. You should see a status table with information about the PHP module.

Unix/OpenBSD installs

This section contains notes and hints specific to installing PHP on OpenBSD (<http://www.openbsd.org/>).

Using Ports

This is the recommended method of installing PHP on OpenBSD, as it will have the latest patches and security fixes applied to it by the maintainers. To use this method, ensure that you have a recent ports tree (<http://www.openbsd.org/ports.html>). Then simply find out which flavors you wish to install, and issue the **make install** command. Below is an example of how to do this.

Example 2-3. OpenBSD Ports Install Example

```
$ cd /usr/ports/www/php4
$ make show VARNAME=FLAVORS
  (choose which flavors you want from the list)
$ env FLAVOR="imap gettext ldap mysql gd" make install
$ /usr/local/sbin/php4-enable
```

Using Packages

There are pre-compiled packages available for your release of OpenBSD (<http://www.openbsd.org/>).

These integrate automatically with the version of Apache installed with the OS. However, since there are a large number of options (called *flavors*) available for this port, you may find it easier to compile it from source using the ports tree. Read the `packages(7)`

(<http://www.openbsd.org/cgi-bin/man.cgi?query=packages>) manual page for more information in what packages are available.

Unix/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.

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 `httpd` 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 2-4. 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.

Installation on Windows systems

This section applies to Windows 95/98/Me and Windows NT/2000/XP. Do not expect PHP to work on 16 bit platforms such as Windows 3.1. Sometimes we refer to the supported Windows platforms as Win32.

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. Also note, that while the InstallShield installer is an easy way to make PHP work, it is restricted in many aspects, as automatic setup of extensions for example is not supported.

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.

Warning

Be aware, that this setup of PHP is not secure. If you would like to have a secure PHP setup, you'd better go on the manual way, and set every option carefully. This automatically working setup gives you an instantly working PHP installation, but it is not meant to be used on online servers.

Manual Installation Steps

This install guide will help you manually install and configure PHP on your Windows webserver. You need to download the zip binary distribution from the downloads page at <http://www.php.net/>. The original version of this guide was compiled by Bob Silva (mailto:bob_silva@mail.umesd.k12.or.us), and 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
- O'Reilly Website Pro
- Xitami
- Netscape Enterprise Server, iPlanet

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.

Warning

The SAPI modules have been significantly improved in the 4.1 release, however, you may find that you encounter possible server errors or other server modules such as ASP failing, in older systems.

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.
- You need to ensure that the DLLs which PHP uses can be found. The precise DLLs involved depend on which web server you use and whether you want to run PHP as a CGI or as a server module. `php4ts.dll` is always used. If you are using a server module (e.g. ISAPI or Apache) then you will need the relevant DLL from the `sapi` folder. If you are using any PHP extension DLLs then you will need those as well. To make sure that the DLLs can be found, you can either copy them to the system directory (e.g. `winnt/system32` or `windows/system`) or you can make sure that they live in the same directory as the main PHP executable or DLL your web server will use (e.g. `php.exe`, `php4apache.dll`).

The PHP binary, the SAPI modules, and some extensions rely on external DLLs for execution. Make sure that these DLLs in the distribution exist in a directory that is in the Windows PATH. The best bet to do it is to copy the files below into your system directory, which is typically:

```
c:\windows\system for Windows 9x/ME
c:\winnt\system32 for Windows NT/2000
c:\windows\system32 for Windows XP
```

The files to copy are:

`php4ts.dll`, if it already exists there, overwrite it

The files in your distribution's 'dlls' directory. If you have them already installed on your system, overwrite them only if

Download the latest version of the Microsoft Data Access Components (MDAC) for your platform, especially if you use Microsoft Windows 9x/NT4. MDAC is available at <http://www.microsoft.com/data/>.

- Copy your chosen ini file (see below) to your '%WINDOWS%' directory on Windows 9x/Me or to your '%SYSTEMROOT%' directory under Windows NT/2000/XP and rename it to `php.ini`. Your '%WINDOWS%' or '%SYSTEMROOT%' directory is typically:

```
c:\windows for Windows 9x/ME/XP
c:\winnt or c:\winnt40 for NT/2000 servers
```

There are two ini files distributed in the zip file, `php.ini-dist` and `php.ini-optimized`. We advise you to use `php.ini-optimized`, because we optimized the default settings in this file for performance, and security. The best is to study all the ini settings and set every element manually yourself. If you would like to achieve the best security, then this is the way for you, although PHP works fine with these default ini files.

- Edit your new `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\extensions`
 - If you are using OmniHTTPd, do not follow the next step. Set the 'doc_root' to point to your webserver's document_root. For example: `c:\apache\htdocs` or `c:\webroot`
 - Choose which extensions you would like to load when PHP starts. See the section about Windows extensions, about how to set up one, and what is already built in. Note that on a new installation it is advisable to first get PHP working and tested without any extensions before enabling them in `php.ini`.
 - On PWS and IIS, you can set the `browscap.ini` to point to:


```
c:\windows\system\inetsrv\browscap.ini on Windows 9x/Me,
c:\winnt\system32\inetsrv\browscap.ini on NT/2000, and
c:\windows\system32\inetsrv\browscap.ini on XP.
```
 - Note that the `mibs` directory supplied with the Windows distribution contains support files for SNMP. This directory should be moved to `DRIVE:\usr\mibs` (DRIVE being the drive where PHP is installed.)
 - If you're using NTFS on Windows NT, 2000 or XP, make sure that the user running the webserver has read permissions to your `php.ini` (e.g. make it readable by Everyone).
- For PWS give execution permission to the webroot:

- Start PWS Web Manager
- Edit Properties of the "Home"-Directory
- Select the "execute"-Checkbox

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/anoncv.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 unzip 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 `vcvars.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 `vcvars.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. You can choose which extensions you would like to load when PHP starts by uncommenting the: `'extension=php_*.dll'` lines in `php.ini`. You can also load a module dynamically in your script using `dl()`.

The DLLs for PHP extensions are prefixed with `'php_'` in PHP 4 (and `'php3_'` in PHP 3). This prevents confusion between PHP extensions and their supporting libraries.

Note: In PHP 4.0.6 BCMath, Calendar, COM, FTP, MySQL, ODBC, 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.

Note: Some of these extensions need extra DLLs to work. Couple of them can be found in the distribution package, in the 'dlls' folder but some, for example Oracle (`php_oci8.dll`) require DLLs which are not bundled with the distribution package.

Copy the bundled DLLs from 'DLLs' folder to your Windows PATH, safe places are:

```
c:\windows\system for Windows 9x/Me
c:\winnt\system32 for Windows NT/2000
c:\windows\system32 for Windows XP
```

If you have them already installed on your system, overwrite them only if something doesn't work correctly (Before overwriting them, it is a good idea to make a backup of them, or move them to another folder - just in case something goes wrong).

Table 2-1. PHP Extensions

Extension	Description	Notes
php_bz2.dll	bzip2 compression functions	None
php_calendar.dll	Calendar conversion functions	Built in since PHP 4.0.3
php_cpdf.dll	ClibPDF functions	None
php_crack.dll	Crack functions	None
php3_crypt.dll	Crypt functions	unknown
php_ctype.dll	ctype family functions	None
php_curl.dll	CURL, Client URL library functions	Requires: libeay32.dll, ssleay32.dll (bundled)
php_cybercash.dll	Cybercash payment functions	None
php_db.dll	DBM functions	Deprecated. Use DBA instead (php_dba.dll)
php_dba.dll	DBA: DataBase (dbm-style) Abstraction layer functions	None
php_dbase.dll	dBase functions	None
php3_dbm.dll	Berkeley DB2 library	unknown
php_domxml.dll	DOM XML functions	Requires: libxml2.dll (bundled)
php_dotnet.dll	.NET functions	None
php_exif.dll	Read EXIF headers from JPEG	None
php_fbsql.dll	FrontBase functions	None
php_fdf.dll	PDF: Forms Data Format functions.	Requires: fdfdk.dll (bundled)
php_filepro.dll	filePro functions	Read-only access
php_ftp.dll	FTP functions	Built-in since PHP 4.0.3
php_gd.dll	GD library image functions	None
php_gettext.dll	Gettext functions	Requires: gnu_gettext.dll (bundled)
php_hyperwave.dll	HyperWave functions	None
php_iconv.dll	ICONV charset conversion	Requires: iconv-1.3.dll (bundled)
php_ifx.dll	Informix functions	Requires: Informix libraries
php_iisfunc.dll	IIS management functions	None
php_imap.dll	IMAP POP3 and NNTP functions	PHP 3: php3_imap4r1.dll
php_ingres.dll	Ingres II functions	Requires: Ingres II libraries
php_interbase.dll	InterBase functions	Requires: gds32.dll (bundled)
php_java.dll	Java extension	Requires: jvm.dll (bundled)
php_ldap.dll	LDAP functions	Requires: libsasl.dll (bundled)
php_mhash.dll	Mhash Functions	None
php_ming.dll	Ming functions for Flash	None
php_mssql.dll	mSQL functions	Requires: mssql.dll (bundled)

Extension	Description	Notes
php3_msql1.dll	mSQL 1 client	unknown
php3_msql2.dll	mSQL 2 client	unknown
php_mssql.dll	MSSQL functions	Requires: ntwdblib.dll (bundled)
php3_mysql.dll	MySQL functions	Built-in in PHP 4
php3_nsmail.dll	Netscape mail functions	unknown
php3_oci73.dll	Oracle functions	unknown
php_oci8.dll	Oracle 8 functions	Requires: Oracle 8 client libraries
php_openssl.dll	OpenSSL functions	Requires: libeay32.dll (bundled)
php_oracle.dll	Oracle functions	Requires: Oracle 7 client libraries
php_pdf.dll	PDF functions	None
php_pgsql.dll	PostgreSQL functions	None
php_printer.dll	Printer functions	None
php_sablot.dll	XSLT functions	Requires: sablot.dll (bundled)
php_snmp.dll	SNMP get and walk functions	NT only!
php_sybase_ct.dll	Sybase functions	Requires: Sybase client libraries
php_yaz.dll	YAZ functions	None
php_zlib.dll	ZLib compression functions	None

Servers-CGI/Commandline

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.

Servers-Apache

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. The version numbers have been omitted here, to ensure the instructions are not incorrect. You will need to replace the 'xxx' here with the correct values from your files.

Example 2-5. Installation Instructions (Apache Shared Module Version) for PHP 4

```
1. gunzip apache_xxx.tar.gz
2. tar -xvf apache_xxx.tar
3. gunzip php-xxx.tar.gz
4. tar -xvf php-xxx.tar
5. cd apache_xxx
6. ./configure --prefix=/www --enable-module=so
7. make
8. make install
9. cd ../php-xxx
10. ./configure --with-mysql --with-apxs=/www/bin/apxs
11. make
12. make install
```

If you decide to change your configure options after installation you only need to repeat the last three steps. You only need to restart apache for the new module to take effect. A recompile of Apache is not needed.

```
11. 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.

```
12. Edit your httpd.conf or srm.conf file and check that these lines are
    present and not commented out:
```

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

```
LoadModule php4_module      libexec/libphp4.so
```

You can choose any extension you wish here. `.php` is simply the one we suggest. You can even include `.html`, and `.php3` can be added for backwards compatibility.

The path on the right hand side of the `LoadModule` statement must point to the path of the PHP module on your system. The above statement is correct for the steps shown above.

13. 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.)

Depending on your Apache install and Unix variant, there are many possible ways to stop and restart the server. Below are some typical lines used in restarting the server, for different apache/unix installations. You should replace `/path/to/` with the path to these applications on your systems.

1. Several Linux and SysV variants:
`/etc/rc.d/init.d/httpd restart`

2. Using `apachectl` scripts:
`/path/to/apachectl stop`
`/path/to/apachectl start`

3. `httpdctl` and `httpsdctl` (Using OpenSSL), similar to `apachectl`:
`/path/to/httpsdctl stop`
`/path/to/httpsdctl start`

4. Using `mod_ssl`, or another SSL server, you may want to manually stop and start:
`/path/to/apachectl stop`
`/path/to/apachectl startssl`

The locations of the `apachectl` and `http(s)dctl` binaries often vary. If your system has `locate` or `whereis` or `which` commands, these can assist you in finding your server control programs.

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 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>).

Note: Apache's default `http.conf` currently ships with a section that looks like this: `User nobody`
`Group #-1` Unless you change that to `"Group ngroup"` or something like that (`"Group daemon"` is also very common) PHP will not be able to open files.

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.

It is worth noting here that now the SAPI module has been made more stable under windows, we recommend it's use above the CGI binary, since it is more transparent and secure.

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 Manual Installation Steps section, you need to insert these lines to your Apache configuration file to set up the CGI binary:

- `ScriptAlias /php/ "c:/php/"`
- `AddType application/x-httpd-php .php .html`
- `Action application/x-httpd-php "/php/php.exe"`

Note that the second line in the list above can be found in the actual versions of `httpd.conf`, but it is commented out. Remember also to substitute the `c:/php/` for your actual path to PHP.

Warning

By using the CGI setup, your server is open to several possible attacks. Please read our CGI security section to learn how to defend yourself from attacks.

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

- `LoadModule php4_module c:/php/sapi/php4apache.dll`
- `AddType application/x-httpd-php .php .html`

After changing the configuration file, remember to restart the server, for example, `NET STOP APACHE` followed by `NET START APACHE`, if you run Apache as a Windows Service, or use your regular shortcuts.

Note: You may find after using the windows installer for Apache that you need to define the `AddModule` directive for `mod_php4.c` in the configuration file (`httpd.conf`). This is done by adding `AddModule mod_php4.c` to the `AddModule` list, near the beginning of the configuration file. This is especially important if the `ClearModuleList` directive is defined. Failure to do this may mean PHP will not be registered as an Apache module.

There are 2 ways you can use the source code highlighting feature, however their ability to work depends on your installation. If you have configured Apache to use PHP as an ISAPI module, then by adding the

following line to your configuration file you can use this feature: `AddType application/x-httpd-php-source .phps`

If you chose to configure Apache to use PHP as a CGI binary, you will need to use the `show_source()` function. To do this simply create a PHP script file and add this code: `<?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.

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

Servers-Caudium

PHP 4 can be built 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 2-6. 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 `<filename>PHP4.so</filename>`. Also make sure that `<filename>caudium/server/lib/[pike-version]/PHP4.so</filename>` 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.

Servers-fhttpd

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.

Servers-IIS/PWS

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 REG file included with the distribution (`pws-php4cgi.reg`). 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. For example `.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`.
- Repeat these steps for each extension you wish to associate with PHP scripts.

The following steps do not affect the web server installation and only apply if you want your php scripts to be executed when they are run from the command line (ex. run `c:\myscripts\test.php`) or by

double clicking on them in a directory viewer window. You may wish to skip these steps as you might prefer the PHP files to load into a text editor when you double click on them.

- 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\ScriptMap] ".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\ScriptMap] ".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/XP and IIS 4 or newer

To install PHP on an NT/2000/XP 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/XP). 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` (assuming that you have unzipped PHP in `c:\php\`).
- 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 may also like to check the 'check that file exists' box - for a small performance penalty, IIS (or PWS) will check that the script file exists and sort out authentication before firing up php. This means that you will get sensible 404 style error messages instead of cgi errors complaining that php did not output any data.

You must start over from the previous step for each extension you want associated with PHP scripts. `.php` and `.html` are common, although `.php3` may be required for legacy applications.

- 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 `I_USR_` 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 (`NET STOP iisadmin`)
- Start IIS again (`NET START w3svc`)

Servers-Netscape and iPlanet

This section contains notes and hints specific to Netscape and iPlanet installs of PHP, both for Sun Solaris and Windows versions.

You can find more information about setting up PHP for the Netscape Enterprise Server here:
<http://benoit.noss.free.fr/php/install-php4.html>

Installing PHP with Netscape on Sun Solaris

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 2-7. 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/netnscape/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 be 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 2-8. 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-httpd-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/netnscape/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=
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-httpd-php"
```

```
.
.
```

```
</Object>
```

```
<Object name="x-httpd-php">
```

```
ObjectType fn="force-type" type="magnus-internal/x-httpd-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>
```

If you are running Netscape Enterprise 4.x, then you should use the following:

Example 2-9. Configuration Example for Netscape Enterprise 4.x

Place these lines after the mime types init, and everything else is similar to the example configuration above.

From: Graeme Hoose (GraemeHoose@BrightStation.com)

```
Init fn="load-modules" shlib="/path/to/server4/bin/libphp4.so" funcs="php4_init,php4_close,p
Init fn="php4_init" LateInit="yes"
```

Installing PHP with Netscape on Windows

To Install PHP as CGI (for Netscape Enterprise Server, iPlanet, perhaps Fastrack), do the following:

- Copy `php4ts.dll` to your systemroot (the directory where you installed windows)
- Make a file association from the command line. Type the following two lines:

```
assoc .php=PHPScript
ftype PHPScript=c:\php\php.exe %1 %*
```

- In the Netscape Enterprise Administration Server create a dummy shellcgi directory and remove it just after (this step creates 5 important lines in obj.conf and allow the web server to handle shellcgi scripts).
- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/shellcgi, File Suffix:php).
- Do it for each web server instance you want php to run

More details about setting up PHP as a CGI executable can be found here:

<http://benoit.noss.free.fr/php/install-php.html>

To Install PHP as NSAPI (for Netscape Enterprise Server, iPlanet, perhaps Fastrack, do the following:

- Copy `php4ts.dll` to your systemroot (the directory where you installed windows)
- Make a file association from the command line. Type the following two lines:

```
assoc .php=PHPScript
ftype PHPScript=c:\php\php.exe %1 %*
```

- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/x-httpd-php, File Suffix:php).
- Stop your web service and edit `obj.conf`. At the end of the Init section, place these two lines (necessarily after mime type init!):

```
Init fn="load-modules" funcs="php4_init,php4_close,php4_execute,php4_auth_trans" shlib="c:
Init fn="php4_init" errorString="Failed to initialise PHP!"
```

- In The `< Object name="default" >` section, place this line necessarily after all 'ObjectType' and before all 'AddLog' lines:

```
Service fn="php4_execute" type="magnus-internal/x-httpd-php"
```

- At the end of the file, create a new object called `x-httpd-php`, by inserting these lines:

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

- Restart your web service and apply changes
- Do it for each web server instance you want PHP to run

More details about setting up PHP as an NSAPI filter can be found here:
<http://benoit.noss.free.fr/php/install-php4.html>

Servers-OmniHTTPd Server

This section contains notes and hints specific to OmniHTTPd.

OmniHTTPd 2.0b1 and up for Windows

You need to complete the following steps to make PHP work with OmniHTTPd. This is a CGI executable setup. SAPI is supported by OmniHTTPd, but some tests have shown that it is not so stable to use PHP as an ISAPI module.

- 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.

Servers-Oreilly Website Pro

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 executable file, enter 'wwwserver/shellcgi', if you choose the ISAPI module, enter 'wwwserver/isapi' (both without quotes).

Servers-Xitami

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.
- Restart the server to reflect changes.

Servers-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 web servers supporting the CGI standard.

Problems?

Read the FAQ

Some problems are more common than others. The most common ones are listed in the PHP FAQ, part of this manual.

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.

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://bugs.php.net/>. Please do not send bug reports in mailing list or personal letters. The bug system is also suitable to submit feature requests.

Read the How to report a bug (<http://bugs.php.net/how-to-report.php>) document before submitting any bug reports!

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 PHP 4 `configure` scripts (as of 4.1.0), used when compiling in Unix-like environments. Some are available in PHP 3, some in PHP 4, and some in both. This is not noted yet.

- Database
- Graphics

- Miscellaneous
- PHP Behaviour
- Server
- XML

Configure Options in PHP 4

Note: These options are only used in PHP 4 as of PHP 4.1.0. Some are available in older versions of PHP 4, some even in PHP 3, some only in PHP 4.1.0. If you want to compile an older version, some options will probably not be available.

Database options

`--with-db`

Include old xDBM support (deprecated).

`--enable-dba=shared`

Build DBA as a shared module.

`--with-gdbm[=DIR]`

Include GDBM support.

`--with-ndbm[=DIR]`

Include NDBM support.

`--with-db2[=DIR]`

Include Berkeley DB2 support.

`--with-db3[=DIR]`

Include Berkeley DB3 support.

`--with-dbm[=DIR]`

Include DBM support.

`--with-cdb[=DIR]`

Include CDB support.

`--enable-dbase`

Enable the bundled dbase library.

`--with-dbplus`
Include dbplus support.

`--enable-dbx`
Enable dbx.

`--with-fbsql[=DIR]`
Include FrontBase support. DIR is the FrontBase base directory.

`--enable-filepro`
Enable the bundled read-only filePro support.

`--with-fribidi[=DIR]`
Include fribidi support (requires FriBidi >=0.1.12). DIR is the fribidi installation directory - default /usr/local/.

`--with-informix[=DIR]`
Include Informix support. DIR is the Informix base install directory, defaults to nothing.

`--with-ingres[=DIR]`
Include Ingres II support. DIR is the Ingres base directory (default /II/ingres).

`--with-interbase[=DIR]`
Include InterBase support. DIR is the InterBase base install directory, defaults to /usr/interbase.

`--with-mysql[=DIR]`
Include mSQL support. DIR is the mSQL base install directory, defaults to /usr/local/Hughes.

`--with-mysql[=DIR]`
Include MySQL support. DIR is the MySQL base directory. If unspecified, the bundled MySQL library will be used.

`--with-oci8[=DIR]`
Include Oracle-oci8 support. Default DIR is ORACLE_HOME.

`--with-adabas[=DIR]`
Include Adabas D support. DIR is the Adabas base install directory, defaults to /usr/local.

`--with-sapdb[=DIR]`
Include SAP DB support. DIR is SAP DB base install directory, defaults to /usr/local.

`--with-solid[=DIR]`
Include Solid support. DIR is the Solid base install directory, defaults to /usr/local/solid.

`--with-ibm-db2[=DIR]`

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

`--with-empress[=DIR]`

Include Empress support. DIR is the Empress base install directory, defaults to \$EMPRESSPATH. From PHP4, this option only supports Empress Version 8.60 and above.

`--with-empress-bcs[=DIR]`

Include Empress Local Access support. DIR is the Empress base install directory, defaults to \$EMPRESSPATH. From PHP4, this option only supports Empress Version 8.60 and above.

`--with-birdstep[=DIR]`

Include Birdstep support. DIR is the Birdstep base install directory, defaults to /usr/local/birdstep.

`--with-custom-odbc[=DIR]`

Include a user defined ODBC support. The DIR is ODBC install base directory, which defaults to /usr/local. Make sure to define CUSTOM_ODBC_LIBS and have some odbc.h in your include dirs. E.g., you should define following for Sybase SQL Anywhere 5.5.00 on QNX, prior to run configure script: CPPFLAGS="-DODBC_QNX -DSQLANY_BUG" LDFLAGS=-lunix CUSTOM_ODBC_LIBS="-ldblib -lodbc".

`--with-iodbc[=DIR]`

Include iODBC support. DIR is the iODBC base install directory, defaults to /usr/local.

`--with-esoob[=DIR]`

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

`--with-unixODBC[=DIR]`

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

`--with-openlink[=DIR]`

Include OpenLink ODBC support. DIR is the OpenLink base install directory, defaults to /usr/local. This is the same as iODBC.

`--with-dbmaker[=DIR]`

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

`--with-oracle[=DIR]`

Include Oracle-oci7 support. Default DIR is ORACLE_HOME.

`--with-ovrimos[=DIR]`

Include Ovrimos SQL Server support. DIR is the Ovrimos' libsqlcli install directory.

`--with-pgsql[=DIR]`

Include PostgreSQL support. DIR is the PostgreSQL base install directory, 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.

`--with-sybase[=DIR]`

Include Sybase-DB support. DIR is the Sybase home directory, defaults to /home/sybase.

`--with-sybase-ct[=DIR]`

Include Sybase-CT support. DIR is the Sybase home directory. Defaults to /home/sybase.

`--disable-unified-odbc`

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

Graphics options

`--with-gd[=DIR]`

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.

`--enable-gd-native-ttf`

GD: Enable TrueType string function in gd.

`--with-jpeg-dir=DIR`

GD: Set the path to libjpeg install prefix.

`--with-png-dir=DIR`

GD: Set the path to libpng install prefix.

`--with-xpm-dir=DIR`

GD: Set the path to libXpm install prefix.

`--with-freetype-dir=DIR`

GD: Set the path to freetype2 install prefix.

`--with-ttf[=DIR]`

GD: Include FreeType 1.x support.

`--with-t1lib[=DIR]`

GD: Include T1lib support.

`--with-cpdfplib[=DIR]`

Include cpdfplib support (requires cpdfplib >= 2). DIR is the cpdfplib install directory, defaults to /usr.

`--with-jpeg-dir[=DIR]`
 jpeg dir for cpdfliB 2.x.

`--with-tiff-dir[=DIR]`
 tiff dir for cpdfliB 2.x.

`--with-pdfliB[=DIR]`
 Include PDFliB support. DIR is the pdfliB base install directory, defaults to /usr/local. Set DIR to shared to build as dl, or shared,DIR to build as dl and still specify DIR.

`--with-jpeg-dir[=DIR]`
 PDFLIB: define libjpeg install directory.

`--with-png-dir[=DIR]`
 PDFLIB: define libpng install directory.

`--with-tiff-dir[=DIR]`
 PDFLIB: define libtiff install directory.

`--with-swf[=DIR]`
 Include swf support.

`--without-gd`
 Disable GD support. PHP 3 only!

`--with-imagick`
 Include ImageMagick support. DIR is the install directory, and if left out, PHP will try to find it on its own. [experimental]. PHP 3 only!

`--with-ming[=DIR]`
 Include ming support.

Misc options

`--enable-force-cgi-redirect`
 Enable the security check for internal server redirects. You should use this if you are running the CGI version with Apache.

`--enable-discard-path`
 If this is enabled, the PHP CGI binary can safely be placed outside of the web tree and people will not be able to circumvent .htaccess security.

`--with-fastcgi=SRCDIR`
 Build PHP as FastCGI application.

`--enable-debug`
Compile with debugging symbols.

`--with-layout=TYPE`
Sets how installed files will be laid out. Type is one of PHP (default) or GNU.

`--with-pear=DIR`
Install PEAR in DIR (default PREFIX/lib/php).

`--without-pear`
Do not install PEAR.

`--with-openssl[=DIR]`
Include OpenSSL support (requires OpenSSL >= 0.9.5).

`--enable-sigchild`
Enable PHP's own SIGCHLD handler.

`--disable-rpath`
Disable passing additional runtime library search paths.

`--enable-libgcc`
Enable explicitly linking against libgcc.

`--enable-dmalloc`
Enable dmalloc.

`--enable-php-streams`
Include experimental php streams. Do not use unless you are testing the code!

`--with-zlib-dir=<DIR>`
Define the location of zlib install directory.

`--with-zlib[=DIR]`
Include zlib support (requires zlib >= 1.0.9). DIR is the zlib install directory.

`--with-aspell[=DIR]`
Include ASPELL support.

`--enable-bcmath`
Enable bc style precision math functions.

`--with-bz2[=DIR]`
Include BZip2 support.

--enable-calendar
Enable support for calendar conversion.

--with-ccvs[=DIR]
Include CCVS support.

--with-crack[=DIR]
Include crack support.

--enable-ctype
Enable ctype support.

--with-curl[=DIR]
Include CURL support.

--with-cybercash[=DIR]
Include CyberCash support. DIR is the CyberCash MCK install directory.

--with-cybermut[=DIR]
Include CyberMut (French Credit Mutuel telepaiement).

--with-cyrus
Include cyrus IMAP support.

--enable-exif
Enable exif support.

--with-fdftk[=DIR]
Include fdftk support.

--enable-ftp
Enable FTP support.

--with-gettext[=DIR]
Include GNU gettext support. DIR is the gettext install directory, defaults to /usr/local.

--with-gmp
Include gmp support.

--with-hyperwave
Include Hyperwave support.

--with-icap[=DIR]
Include ICAP support.

`--with-iconv[=DIR]`

Include iconv support.

`--with-imap[=DIR]`

Include IMAP support. DIR is the c-client install prefix.

`--with-kerberos[=DIR]`

IMAP: Include Kerberos support. DIR is the Kerberos install dir.

`--with-imap-ssl[=DIR]`

IMAP: Include SSL support. DIR is the OpenSSL install dir.

`--with-ircg-config`

Path to the ircg-config script.

`--with-ircg`

Include ircg support.

`--with-java[=DIR]`

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

`--with-ldap[=DIR]`

Include LDAP support. DIR is the LDAP base install directory.

`--enable-mailparse`

Enable mailparse support.

`--enable-mbstring`

Enable multibyte string support.

`--enable-mbstr-enc-trans`

Enable japanese encoding translation.

`--with-mcal[=DIR]`

Include MCAL support.

`--with-mcrypt[=DIR]`

Include mcrypt support. DIR is the mcrypt install directory.

`--with-mhash[=DIR]`

Include mhash support. DIR is the mhash install directory.

```

--with-mnogosearch[=DIR]
    Include mnoGoSearch support. DIR is the mnoGoSearch base install directory, defaults to
    /usr/local/mnogosearch.

--with-muscat[=DIR]
    Include muscat support.

--with-ncurses
    Include ncurses support.

--enable-pcntl
    Enable experimental pcntl support (CGI ONLY!)

--without-pcre-regex
    Do not include Perl Compatible Regular Expressions support. Use --with-pcre-regex=DIR to
    specify DIR where PCRE's include and library files are located, if not using bundled library.

--with-pfpro[=DIR]
    Include Verisign Payflow Pro support.

--disable-posix
    Disable POSIX-like functions.

--with-ispell[=DIR]
    Include ISPELL support.

--with-qtdom
    Include QtDOM support (requires Qt >= 2.2.0).

--with-libedit[=DIR]
    Include libedit readline replacement.

--with-readline[=DIR]
    Include readline support. DIR is the readline install directory.

--with-recode[=DIR]
    Include recode support. DIR is the recode install directory.

--with-satellite[=DIR]
    Enable CORBA support via Satellite (EXPERIMENTAL) DIR is the base directory for ORBit.

--with-mm[=DIR]
    Include mm support for session storage.

```

`--enable-trans-sid`
 Enable transparent session id propagation.

`--disable-session`
 Disable session support.

`--enable-shmop`
 Enable shmop support.

`--with-snmp[=DIR]`
 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`
 Enable UCD SNMP hack.

`--enable-sockets`
 Enable sockets support.

`--with-regex=TYPE`
 regex library type: system, apache, php.

`--with-system-regex`
 Use system regex library (deprecated).

`--enable-sysvsem`
 Enable System V semaphore support.

`--enable-sysvshm`
 Enable the System V shared memory support.

`--with-vpopmail[=DIR]`
 Include vpopmail support.

`--with-tsrm-pthreads`
 Use POSIX threads (default).

`--enable-shared[=PKGS]`
 Build shared libraries [default=yes].

`--enable-static[=PKGS]`
 Build static libraries [default=yes].

`--enable-fast-install[=PKGS]`
 Optimize for fast installation [default=yes].

`--with-gnu-ld`
 Assume the C compiler uses GNU ld [default=no].

`--disable-libtool-lock`
 Avoid locking (might break parallel builds).

`--with-pic`
 Try to use only PIC/non-PIC objects [default=use both].

`--with-yaz[=DIR]`
 Include YAZ support (ANSI/NISO Z39.50). *DIR* is the YAZ bin install directory.

`--enable-memory-limit`
 Compile with memory limit support.

`--disable-url-fopen-wrapper`
 Disable the URL-aware fopen wrapper that allows accessing files via HTTP or FTP.

`--enable-versioning`
 Export only required symbols. See `INSTALL` for more information.

`--disable-bcmath`
 Compile without bc style precision math functions. PHP 3 only!

`--with-imsp[=DIR]`
 Include IMSp support (*DIR* is IMSP's include dir and libimsp.a dir). PHP 3 only!

`--with-ftp`
 Include FTP support. PHP 3 only!

`--with-mck[=DIR]`
 Include Cybercash MCK support. *DIR* is the cybercash mck build directory, defaults to `/usr/src/mck-3.2.0.3-linux` for help look in `extra/cyberlib`. PHP 3 only!

`--disable-overload`
 Disable user-space object overloading support. PHP 3 only!

`--enable-yp`
 Include YP support. PHP 3 only!

`--with-zip`
 Include ZIP support (requires `zziplib >= 0.10.6`). PHP 3 only!

`--with-mod-dav=DIR`

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

`--enable-debugger`

Compile with remote debugging functions. PHP 3 only!

`--enable-versioning`

Take advantage of versioning and scoping provided by Solaris 2.x and Linux. PHP 3 only!

PHP options

`--enable-maintainer-mode`

Enable make rules and dependencies not useful (and sometimes confusing) to the casual installer.

`--with-config-file-path=PATH`

Sets the path in which to look for `php.ini`, defaults to `PREFIX/lib`.

`--enable-safe-mode`

Enable safe mode by default.

`--with-exec-dir[=DIR]`

Only allow executables in DIR when in safe mode defaults to `/usr/local/php/bin`.

`--enable-magic-quotes`

Enable magic quotes by default.

`--disable-short-tags`

Disable the short-form `<? start` tag by default.

Server options

`--with-aolserver=DIR`

Specify path to the installed AOLserver.

`--with-apxs[=FILE]`

Build shared Apache module. FILE is the optional pathname to the Apache apxs tool; defaults to `apxs`.

`--with-apache[=DIR]`

Build Apache module. DIR is the top-level Apache build directory, defaults to `/usr/local/apache`.

`--with-mod_charset`

Enable transfer tables for mod_charset (Rus Apache).

`--with-apxs2[=FILE]`

Build shared Apache 2.0 module. FILE is the optional pathname to the Apache apxs tool; defaults to apxs.

`--with-fhttpd[=DIR]`

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

`--with-isapi=DIR`

Build PHP as an ISAPI module for use with Zeus.

`--with-nsapi=DIR`

Specify path to the installed Netscape Server.

`--with-phhttpd=DIR`

No information yet.

`--with-pi3web=DIR`

Build PHP as a module for use with Pi3Web.

`--with-roxen=DIR`

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

`--enable-roxen-zts`

Build the Roxen module using Zend Thread Safety.

`--with-servlet[=DIR]`

Include servlet support. DIR is the base install directory for the JSDK. This SAPI prereqs the java extension must be built as a shared dl.

`--with-thttpd=SRCDIR`

Build PHP as thttpd module.

`--with-tux=MODULEDIR`

Build PHP as a TUX module (Linux only).

XML options

`--with-dom[=DIR]`

Include DOM support (requires libxml >= 2.4.2). DIR is the libxml install directory, defaults to /usr.

`--disable-xml`

Disable XML support using bundled expat lib.

`--with-expat-dir=DIR`

XML: external libexpat install dir.

`--with-xmlrpc[=DIR]`

Include XMLRPC-EPI support.

`--enable-wddx`

Enable WDDX support.

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.

Example 3-1. `php.ini` example

```
; any text on a line after an unquoted semicolon (;) is ignored
[php] ; section markers (text within square brackets) are also ignored
; Boolean values can be set to either:
;   true, on, yes
; or false, off, no, none
register_globals = off
magic_quotes_gpc = yes

; you can enclose strings in double-quotes
include_path = "./usr/local/lib/php"

; backslashes are treated the same as any other character
include_path = ".;c:\php\lib"
```

When using PHP as an Apache module, you can also change the configuration settings using directives in Apache configuration files and `.htaccess` files (You will need "AllowOverride Options" or "AllowOverride All" privileges)

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 several Apache directives that allow you to change the PHP configuration from within the Apache configuration file itself.

`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.

Example 3-2. Apache configuration example

```

<IfModule mod_php4.c>
  php_value include_path ".:usr/local/lib/php"
  php_flag safe_mode on
</IfModule>
<IfModule mod_php3.c>
  php3_include_path ".:usr/local/lib/php"
  php3_safe_mode on
</IfModule>

```

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 `--d isable-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-prepend.

cgi_ext string

display_errors boolean

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

doc_root string

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

engine boolean

This directive is really only useful in the Apache module version of PHP. It is used by sites that would like to turn PHP parsing on and off on a per-directory or per-virtual server basis. By putting **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).

html_errors boolean

Turn off HTML tags in error messages.

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.

The restriction specified with `open_basedir` is actually a prefix, not a directory name. This means that `"open_basedir = /dir/incl"` also allows access to `"/dir/include"` and `"/dir/incls"` if they exist. When you want to restrict access to only the specified directory, end with a slash. For example: `"open_basedir = /dir/incl/"`

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.

Note, that this option is not available in PHP 4. Use `variables_order` instead.

variables_order string

Set the order of the EGPCS (Environment, GET, POST, Cookie, Server) variable parsing. The default setting of this directive is "EGPCS". Setting this to "GP", for example, will cause PHP to completely ignore environment variables, cookies and server variables, and to overwrite any GET method variables with POST-method variables of the same name.

See also `register_globals`.

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 3-3. UNIX `include_path`

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

Example 3-4. 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 run before it is terminated by the parser. This helps prevent poorly written scripts from tying up the server. The default setting is 30.

The maximum execution time is not affected by system calls, the sleep() function, etc. Please see the set_time_limit() function for more details.

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

precision integer

The number of significant digits displayed in floating point numbers.

register_argc_argv boolean

Tells PHP whether to declare the argv & argc variables (that would contain the GET information).

See also command line. Also, this directive became available in PHP 4.0.0 and was always "on" before that.

post_max_size integer

Sets max size of post data allowed. This setting also affects file upload. To upload large files, this value must be larger than upload_max_filesize.

If memory limit is enabled by configure script, memory_limit also affects file uploading. Generally speaking, memory_limit should be larger than post_max_size.

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 \$_ENV, \$_GET, \$_POST, \$_COOKIE, and \$_SERVER arrays in the global scope.

Please note that you need to set AllowOverride All in your Directory block in the apache config file for this to work.

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 `$_ENV`, `$_GET`, `$_POST`, `$_COOKIE`, and `$_SERVER`.

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.

upload_max_filesize integer

The maximum size of an uploaded file. The value is in bytes.

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 (.

Safe Mode Configuration Directives

safe_mode boolean

Whether to enable PHP's safe mode. Read the Security and Safe Mode chapters 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.

mSQL Configuration Directives

mysql.allow_persistent boolean

Whether to allow persistent mSQL connections.

mysql.max_persistent integer

The maximum number of persistent mSQL connections per process.

mysql.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()`.

Multi-Byte String Configuration Directives

mbstring.internal_encoding string

`mbstring.internal_encoding` defines default internal character encoding.

mbstring.http_input string

`mbstring.http_input` defines default HTTP input character encoding.

mbstring.http_output string

`mbstring.http_output` defines default HTTP output character encoding.

mbstring.detect_order string

`mbstring.detect_order` defines default character encoding detection order.

mbstring.substitute_character string

`mbstring.substitute_character` defines character to substitute for invalid character codes.

Exif Configuration Directives

Exif supports automatic conversion for Unicode and JIS character encodings of user comments when module `mbstring` is available. This is done by first decoding the comment using the specified character set. The result is then encoded with another character set which should match your http output.

exif.encode_unicode string

`exif.encode_unicode` defines the character set UNICOD user comments are handled. This defaults to ISO-8859-15 which should work for most non asian countries. The setting can be empty or must be an encoding supported by `mbstring`. If it is empty the current internal encoding of `mbstring` is used.

exif.decode_unicode_motorola string

`exif.decode_unicode_motorola` defines the image internal character set for Unicode encoded user comments if image is in motorola byte order (big-endian). This setting cannot be empty but you can specify a list of encodings supported by mbstring. The default is UCS-2BE.

exif.decode_unicode_intel string

`exif.decode_unicode_intel` defines the image internal character set for Unicode encoded user comments if image is in intel byte order (little-endian). This setting cannot be empty but you can specify a list of encodings supported by mbstring. The default is UCS-2LE.

exif.encode_jis string

`exif.encode_jis` defines the character set JIS user comments are handled. This defaults to an empty value which forces the functions to use the current internal encoding of mbstring.

exif.decode_jis_motorola string

`exif.decode_jis_motorola` defines the image internal character set for JIS encoded user comments if image is in motorola byte order (big-endian). This setting cannot be empty but you can specify a list of encodings supported by mbstring. The default is JIS.

exif.decode_jis_intel string

`exif.decode_jis_intel` defines the image internal character set for JIS encoded user comments if image is in intel byte order (little-endian). This setting cannot be empty but you can specify a list of encodings supported by mbstring. The default is JIS.

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 with some general security advice, explains the different configuration option combinations and the situations they can be safely used, and describes different considerations in coding for different levels of security.

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.

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 attached to it, it is discovered that PHP is now prevented from writing any files to user directories. Or perhaps it has been prevented from accessing or changing databases. It has equally been secured from writing good and bad files, or entering good and bad database transactions.

A frequent security mistake made at this point is to allow apache root permissions, or to escalate apache's abilities in some other way.

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.

There are some simpler solutions. By using `open_basedir` you can control and restrict what directories are allowed to be used for PHP. You can also set up apache-only areas, to restrict all web based activity to non-user, or non-system, files.

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/passwd, 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 = $_HTTP_POST_VARS['user_submitted_name'];
$homedir = "/home/$username";
$file_to_delete = "$userfile";
unlink ($homedir/$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_SERVER_VARS['REMOTE_USER']; // using an authentication mechanism

$home_dir = "/home/$username";

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

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

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

However, even this is not without its flaws. If your authentication system allowed users to create their own user logins, and a user chose the login "../etc/", the system is once again exposed. For this reason, you may prefer to write a more customized check:

Example 4-4. More secure file name checking

```
<?php
$username = $_HTTP_SERVER_VARS['REMOTE_USER']; // using an authentication mechanism
$home_dir = "/home/$username";

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

if (!ereg('^[^./][^/]*$', $username))
    die('bad username'); //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.

Database Security

Nowadays, databases are cardinal components of any web based application by enabling websites to provide varying dynamic content. Since very sensitive or secret informations can be stored in such database, you should strongly consider to protect them somehow.

To retrieve or to store any information you need to connect to the database, send a legitimate query, fetch the result, and close the connection. Nowadays, the commonly used query language in this interaction is the Structured Query Language (SQL). See how an attacker can tamper with an SQL query.

As you can realize, PHP cannot protect your database by itself. The following sections aim to be an introduction into the very basics of how to access and manipulate databases within PHP scripts.

Keep in mind this simple rule: defence in depth. In the more place you take the more action to increase the protection of your database, the less probability of that an attacker succeeds, and exposes or abuse any stored secret information. Good design of the database schema and the application deals with your greatest fears.

Designing Databases

The first step is always to create the database, unless you want to use an existing third party's one. When a database is created, it is assigned to an owner, who executed the creation statement. Usually, only the owner (or a superuser) can do anything with the objects in that database, and in order to allow other users to use it, privileges must be granted.

Applications should never connect to the database as its owner or a superuser, because these users can execute any query at will, for example, modifying the schema (e.g. dropping tables) or deleting its entire content.

You may create different database users for every aspect of your application with very limited rights to database objects. The most required privileges should be granted only, and avoid that the same user can interact with the database in different use cases. This means that if intruders gain access to your database using one of these credentials, they can only effect as many changes as your application can.

You are encouraged not to implement all the business logic in the web application (i.e. your script), instead to do it in the database schema using views, triggers or rules. If the system evolves, new ports will be intended to open to the database, and you have to reimplement the logic in each separate database client. Over and above, triggers can be used to transparently and automatically handle fields, which often provides insight when debugging problems with your application or tracing back transactions.

Connecting to Database

You may want to establish the connections over SSL to encrypt client/server communications for

increased security, or you can use ssh to encrypt the network connection between clients and the database server. If either of them is done, then monitoring your traffic and gaining informations in this way will be a hard work.

Encrypted Storage Model

SSL/SSH protects data travelling from the client to the server, SSL/SSH does not protect the persistent data stored in a database. SSL is an on-the-wire protocol.

Once an attacker gains access to your database directly (bypassing the webserver), the stored sensitive data may be exposed or misused, unless the information is protected by the database itself. Encrypting the data is a good way to mitigate this threat, but very few databases offer this type of data encryption.

The easiest way to work around this problem is to first create your own encryption package, and then use it from within your PHP scripts. PHP can assist you in this case with its several extensions, such as Mcrypt and Mhash, covering a wide variety of encryption algorithms. The script encrypts the data before stored first, and decrypts it when retrieving. See the references for further examples how encryption works.

In case of truly hidden data, if its raw representation is not needed (i.e. not be displayed), hashing may be also taken into consideration. The well-known example for the hashing is storing the MD5 hash of a password in a database, instead of the password itself. See also `crypt()` and `md5()`.

Example 4-5. Using hashed password field

```
// storing password hash
$query = sprintf("INSERT INTO users(name,pwd) VALUES('%s','%s');",
                addslashes($username), md5($password));
$result = pg_exec($connection, $query);

// querying if user submitted the right password
$query = sprintf("SELECT 1 FROM users WHERE name='%s' AND pwd='%s';",
                addslashes($username), md5($password));
$result = pg_exec($connection, $query);

if (pg_numrows($result) > 0) {
    echo "Welcome, $username!";
}
else {
    echo "Authentication failed for $username.";
}
```

SQL Injection

Many web developers are unaware of how SQL queries can be tampered with, and assume that an SQL query is a trusted command. It means that SQL queries are able to circumvent access controls, thereby

bypassing standard authentication and authorization checks, and sometimes SQL queries even may allow access to host operating system level commands.

Direct SQL Command Injection is a technique where an attacker creates or alters existing SQL commands to expose hidden data, or to override valuable ones, or even to execute dangerous system level commands on the database host. This is accomplished by the application taking user input and combining it with static parameters to build a SQL query. The following examples are based on true stories, unfortunately.

Owing to the lack of input validation and connecting to the database on behalf of a superuser or the one who can create users, the attacker may create a superuser in your database.

Example 4-6. Splitting the result set into pages ... and making superusers (PostgreSQL and MySQL)

```
$offset = argv[0]; // beware, no input validation!
$query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";
// with PostgreSQL
$result = pg_exec($conn, $query);
// with MySQL
$result = mysql_query($query);
```

Normal users click on the 'next', 'prev' links where the \$offset is encoded into the URL. The script expects that the incoming \$offset is decimal number. However, someone tries to break in with appending urlencode()'d form of the following to the URL

```
// in case of PostgreSQL
0;
insert into pg_shadow(username,usesysid,usesuper,usecatupd,passwd)
    select 'crack', usesysid, 't','t','crack'
    from pg_shadow where username='postgres';
--

// in case of MySQL
0;
UPDATE user SET Password=PASSWORD('crack') WHERE user='root';
FLUSH PRIVILEGES;
```

If it happened, then the script would present a superuser access to him. Note that 0; is to supply a valid offset to the original query and to terminate it.

Note: It is common technique to force the SQL parser to ignore the rest of the query written by the developer with -- which is the comment sign in SQL.

A feasible way to gain passwords is to circumvent your search result pages. What the attacker needs only is to try if there is any submitted variable used in SQL statement which is not handled properly. These filters can be set commonly in a preceding form to customize WHERE, ORDER BY, LIMIT and OFFSET clauses in SELECT statements. If your database supports the UNION construct, the attacker may try to append an entire query to the original one to list passwords from an arbitrary table. Using encrypted password fields is strongly encouraged.

Example 4-7. Listing out articles ... and some passwords (any database server)

```
$query = "SELECT id, name, inserted, size FROM products
          WHERE size = '$size'
          ORDER BY $order LIMIT $limit, $offset;";
$result = odbc_exec($conn, $query);
```

The static part of the query can be combined with another SELECT statement which reveals all passwords:

```
,
union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0' from usertable;
--
```

If this query (playing with the ' and --) were assigned to one of the variables used in \$query, the query beast awakened.

SQL UPDATES are also subject to attacking your database. These queries are also threatened by chopping and appending an entirely new query to it. But the attacker might fiddle with the SET clause. In this case some schema information must be possessed to manipulate the query successfully. This can be acquired by examining the form variable names, or just simply brute forcing. There are not so many naming convention for fields storing passwords or usernames.

Example 4-8. From resetting a password ... to gaining more privileges (any database server)

```
$query = "UPDATE usertable SET pwd='$pwd' WHERE uid='$uid';";
```

But a malicious user submits the value ' or uid like '%admin%'; -- to \$uid to change the admin's password, or simply sets \$pwd to "hehehe", admin='yes', trusted=100 " (with a trailing space) to gain more privileges. Then, the query will be twisted:

```
// $uid == ' or uid like '%admin%'; --
```

```
$query = "UPDATE usertable SET pwd='...' WHERE uid=" or uid like '%admin%'; --
";

// $pwd == "hehehe", admin='yes', trusted=100 "
$query = "UPDATE usertable SET pwd='hehehe', admin='yes', trusted=100 WHERE ...;"
```

A frightening example how operating system level commands can be accessed on some database hosts.

Example 4-9. Attacking the database host's operating system (MSSQL Server)

```
$query = "SELECT * FROM products WHERE id LIKE '%$prod%'";
$result = mssql_query($query);
```

If attacker submits the value `a%' exec master..xp_cmdshell 'net user test testpass /ADD' --` to `$prod`, then the `$query` will be:

```
$query = "SELECT * FROM products
          WHERE id LIKE '%a%'
          exec master..xp_cmdshell 'net user test testpass /ADD'--";
$result = mssql_query($query);
```

MSSQL Server executes the SQL statements in the batch including a command to add a new user to the local accounts database. If this application were running as `sa` and the `MSSQLSERVER` service is running with sufficient privileges, the attacker would now have an account with which to access this machine.

Note: Some of the examples above is tied to a specific database server. This does not mean that a similar attack is impossible against other products. Your database server may be so vulnerable in other manner.

Avoiding techniques

You may plead that the attacker must possess a piece of information about the database schema in most examples. You are right, but you never know when and how it can be taken out, and if it happens, your database may be exposed. If you are using an open source, or publicly available database handling package, which may belong to a content management system or forum, the intruders easily produce a copy of a piece of your code. It may be also a security risk if it is a poorly designed one.

These attacks are mainly based on exploiting the code not being written with security in mind. Never trust on any kind of input, especially which comes from the client side, even though it comes from a

select box, a hidden input field or a cookie. The first example shows that such a blameless query can cause disasters.

- Never connect to the database as a superuser or as the database owner. Use always customized users with very limited privileges.
- Check if the given input has the expected data type. PHP has a wide range of input validating functions, from the simplest ones found in Variable Functions and in Character Type Functions (e.g. `is_numeric()`, `ctype_digit()` respectively) onwards the Perl compatible Regular Expressions support.
- If the application waits for numerical input, consider to verify data with `is_numeric()`, or silently change its type using `settype()`, or use its numeric representation by `sprintf()`.

Example 4-10. A more secure way to compose a query for paging

```
settype($offset, 'integer');
$query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";

// please note %d in the format string, using %s would be meaningless
$query = sprintf("SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET %d;",
                $offset);
```

- Quote each non numeric user input which is passed to the database with `addslashes()` or `addcslashes()`. See the first example. As the examples shows, quotes burnt into the static part of the query is not enough, and can be easily hacked.
- Do not print out any database specific information, especially about the schema, by fair means or foul. See also Error Reporting and Error Handling and Logging Functions.
- You may use stored procedures and previously defined cursors to abstract data access so that users do not directly access tables or views, but this solution has another impacts.

Besides these, you benefit from logging queries either within your script or by the database itself, if it supports. Obviously, the logging is unable to prevent any harmful attempt, but it can be helpful to trace back which application has been circumvented. The log is not useful by itself, but through the information it contains. The more detail is generally better.

Error Reporting

With PHP security, there are two sides to error reporting. One is beneficial to increasing security, the other is detrimental.

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. For example, if an attacker had gleaned

information about a page based on a prior form submission, they may attempt to override variables, or modify them:

Example 4-11. Attacking Variables with a custom HTML page

```
<form method="post" action="attacktarget?username=badfoo&password=badfoo">
<input type="hidden" name="username" value="badfoo">
<input type="hidden" name="password" value="badfoo">
</form>
```

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. Especially dangerous is running code from known sources with built-in debugging handlers, or using common debugging techniques. If the attacker can determine what general technique you are using, they may try to brute-force a page, by sending various common debugging strings:

Example 4-12. Exploiting common debugging variables

```
<form method="post" action="attacktarget?errors=Y&showerrors=1&debug=1">
<input type="hidden" name="errors" value="Y">
<input type="hidden" name="showerrors" value="1">
<input type="hidden" name="debug" value="1">
</form>
```

Regardless of the method of error handling, the ability to probe a system for errors leads to providing an attacker with more information.

For example, the very style of a generic PHP 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.

One way of catching this issue ahead of time is to make use of PHP's own `error_reporting()`, to help you secure your code and find variable usage that may be dangerous. By testing your code, prior to deployment, with `E_ALL`, you can quickly find areas where your variables may be open to poisoning or modification in other ways. Once you are ready for deployment, by using `E_NONE`, you insulate your code from probing.

Example 4-13. Finding dangerous variables with `E_ALL`

```
<?php
if ($username) { // Not initialized or checked before usage
    $good_login = 1;
}
if ($good_login == 1) { // If above test fails, not initialized or checked before usage
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```

Using Register Globals

One feature of PHP that can be used to enhance security is configuring PHP with `register_globals = off`. By turning off the ability for any user-submitted variable to be injected into PHP code, you can reduce the amount of variable poisoning a potential attacker may inflict. They will have to take the additional time to forge submissions, and your internal variables are effectively isolated from user submitted data.

While it does slightly increase the amount of effort required to work with PHP, it has been argued that the benefits far outweigh the effort.

Example 4-14. Working without `register_globals=off`

```
<?php
if ($username) { // can be forged by a user in get/post/cookies
    $good_login = 1;
}

if ($good_login == 1) { // can be forged by a user in get/post/cookies,
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```


Example 4-15. Working with register_globals = off

```
<?php
if($HTTP_COOKIE_VARS['username']){
    // can only come from a cookie, forged or otherwise
    $good_login = 1;
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```

By using this wisely, it's even possible to take preventative measures to warn when forging is being attempted. If you know ahead of time exactly where a variable should be coming from, you can check to see if submitted data is coming from an inappropriate kind of submission. While it doesn't guarantee that data has not been forged, it does require an attacker to guess the right kind of forging.

Example 4-16. Detecting simple variable poisoning

```
<?php
if ($HTTP_COOKIE_VARS['username'] &&
    !$HTTP_POST_VARS['username'] &&
    !$HTTP_GET_VARS['username'] ) {
    // Perform other checks to validate the user name...
    $good_login = 1;
    fpassthru ("/highly/sensitive/data/index.html");
} else {
    mail("admin@example.com", "Possible breakin attempt", $HTTP_SERVER_VARS['REMOTE_ADDR']);
    echo "Security violation, admin has been alerted.";
    exit;
}
?>
```

Of course, simply turning off register_globals does not mean code is secure. For every piece of data that is submitted, it should also be checked in other ways.

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 4-17. 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 an /etc/passwd entry?
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).

Hiding PHP

In general, security by obscurity is one of the weakest forms of security. But in some cases, every little bit of extra security is desirable.

A few simple techniques can help to hide PHP, possibly slowing down an attacker who is attempting to discover weaknesses in your system. By setting `expose_php = off` in your `php.ini` file, you reduce the amount of information available to them.

Another tactic is to configure web servers such as apache to parse different filetypes through PHP, either with an `.htaccess` directive, or in the apache configuration file itself. You can then use misleading file extensions:

Example 4-18. Hiding PHP as another language

```
# Make PHP code look like other code types
AddType application/x-httpd-php .asp .py .pl
```

Or obscure it completely:

Example 4-19. Using unknown types for PHP extensions

```
# Make PHP code look like unknown types
AddType application/x-httpd-php .bop .foo .133t
```

Or hide it as html code, which has a slight performance hit because all html will be parsed through the PHP engine:

Example 4-20. Using html types for PHP extensions

```
# Make all PHP code look like html
AddType application/x-httpd-php .htm .html
```

For this to work effectively, you must rename your PHP files with the above extensions. While it is a form of security through obscurity, it's a minor preventative measure with few drawbacks.

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

When PHP parses a file, it simply passes the text of the file through until it encounters one of the special tags which tell it to start interpreting the text as PHP code. The parser then executes all the code it finds, up until it runs into a PHP closing tag, which tells the parser to just start passing the text through again. This is the mechanism which allows you to embed PHP code inside HTML: everything outside the PHP tags is left utterly alone, while everything inside is parsed as code.

There are four sets of tags which can be used to denote blocks of PHP code. Of these, only two (`<?php. .?>` and `<script language="php">. .</script>`) are always available; the others can be turned on or off from the `php.ini` configuration file. While the short-form tags and ASP-style tags may be convenient, they are not as portable as the longer versions. Also, if you intend to embed PHP code in XML or XHTML, you will need to use the `<?php. .?>` form to conform to the XML.

The tags supported by PHP are:

Example 5-1. Ways of escaping from HTML

1. `<?php echo("if you want to serve XHTML or XML documents, do like this\n"); ?>`
2. `<? echo ("this is the simplest, an SGML processing instruction\n"); ?>`
`<?= expression ?>` This is a shortcut for `"<? echo expression ?>"`
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 the generally preferred method, as it allows for the the use of PHP in XML-conformant code such as XHTML.

The second way is only available if short tags have been enabled. This can be done via the `short_tags()` function (PHP 3 only), by enabling the `short_open_tag` configuration setting in the PHP config file, or by compiling PHP with the `--enable-short-tags` option to **configure**.

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

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

The closing tag for the block will include the immediately trailing newline if one is present. Also, the closing tag automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block.

PHP allows you to use structures like this:

Example 5-2. Advanced escaping

```
<?php
if ($expression) {
    ?>
    <strong>This is true.</strong>
    <?php
} else {
    ?>
    <strong>This is false.</strong>
    <?php
}
?>
```

This works as expected, because when PHP hits the `?>` closing tags, it simply starts outputting whatever it finds until it hits another opening tag. The example given here is contrived, of course, but for outputting large blocks of text, dropping out of PHP parsing mode is generally more efficient than sending all of the text through `echo()` or `print()` or `somesuch`.

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

Introduction

PHP supports eight primitive types.

Four scalar types:

- boolean
- integer
- floating-point number (float)
- string

Two compound types:

- array
- object

And finally two special types:

- resource
- NULL

Note: In this manual you'll often find `mixed` parameters. This pseudo-type indicates multiple possibilities for that parameter.

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.

Note: If you want to check out the type and value of a certain expression, use `var_dump()`.

If you simply want a human-readable representation of the type for debugging, use `gettype()`. To check for a certain type, do *not* use `gettype()`, but use the `is_type` functions.

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](#).

Booleans

This is the easiest type. A boolean expresses a truth value. It can be either `TRUE` or `FALSE`.

Note: The boolean type was introduced in PHP 4.

Syntax

To specify a boolean literal, use either the keyword `TRUE` or `FALSE`. Both are case-insensitive.

```
$foo = True; // assign the value TRUE to $foo
```

Usually you use some kind of operator which returns a boolean value, and then pass it on to a control structure.

```
// == is an operator which returns a boolean
if ($action == "show_version") {
    echo "The version is 1.23";
}

// this is not necessary:
if ($show_separators == TRUE) {
    echo "<hr>\n";
}

// because you can simply type this:
if ($show_separators) {
    echo "<hr>\n";
}
```

Converting to boolean

To explicitly convert a value to boolean, use either the `(bool)` or the `(boolean)` cast. However, in most cases you do not need to use the cast, since a value will be automatically converted if an operator, function or control structure requires a boolean argument.

See also [Type Juggling](#).

When converting to boolean, the following values are considered `FALSE`:

- the boolean `FALSE`
- the integer `0` (zero)
- the float `0.0` (zero)

- the empty string, and the string "0"
 - an array with zero elements
 - an object with zero elements
 - the special type NULL (including unset variables)
- Every other value is considered TRUE (including any resource).

Warning

-1 is considered TRUE, like any other non-zero (whether negative or positive) number!

Integers

An integer is a number of the set $Z = \{ \dots, -2, -1, 0, 1, 2, \dots \}$.

See also: Arbitrary length integers and Floating point numbers

Syntax

Integers can be specified in decimal (10-based), hexadecimal (16-based) or octal (8-based) notation, optionally preceded by a sign (- or +).

If you use the octal notation, you must precede the number with a 0 (zero), to use hexadecimal notation precede the number with 0x.

Example 6-1. Integer literals

```
$a = 1234; # decimal number
$a = -123; # a negative number
$a = 0123; # octal number (equivalent to 83 decimal)
$a = 0x1A; # hexadecimal number (equivalent to 26 decimal)
```

The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed). PHP does not support unsigned integers.

Integer overflow

If you specify a number beyond the bounds of the integer type, it will be interpreted as a float instead. Also, if you perform an operation that results in a number beyond the bounds of the integer type, a float will be returned instead.

```
$large_number = 2147483647;
var_dump($large_number);
// output: int(2147483647)

$large_number = 2147483648;
var_dump($large_number);
// output: float(2147483648)

// this goes also for hexadecimal specified integers:
var_dump( 0x80000000 );
// output: float(2147483648)

$million = 1000000;
$large_number = 50000 * $million;
var_dump($large_number);
// output: float(50000000000)
```

Warning

Unfortunately, there was a bug in PHP so that this does not always work correctly when there are negative numbers involved. For example: when you do `-50000 * $million`, the result will be `-429496728`. However, when both operands are positive there is no problem.

This is solved in PHP 4.1.0.

There is no integer division operator in PHP. `1/2` yields the float `0.5`.

```
var_dump( 25/7 );
// output: float(3.5714285714286)
```

Converting to integer

To explicitly convert a value to integer, use either the `(int)` or the `(integer)` cast. However, in most cases you do not need to use the cast, since a value will be automatically converted if an operator, function or control structure requires a integer argument.

See also type-juggling.

From booleans

`FALSE` will yield 0 (zero), and `TRUE` will yield 1 (one).

From floating point numbers

When converting from float to integer, the number will be rounded *towards zero*.

If the float is beyond the boundaries of integer (usually $\pm 2.15e+9 = 2^{31}$), the result is undefined, since the float hasn't got enough precision to give an exact integer result. No warning, not even a notice will be issued in this case!

Warning

Never cast an unknown fraction to integer, as this can sometimes lead to unexpected results.

```
echo (int) ( (0.1+0.7) * 10 ); // echoes 7!
```

See for more information the warning about float-precision.

From strings

See String conversion

From other types

Caution

Behaviour of converting to integer is undefined for other types. Currently, the behaviour is the same as if the value was first converted to boolean. However, do *not* rely on this behaviour, as it can change without notice.

Floating point numbers

Floating point numbers (AKA "floats", "doubles" or "real numbers") can be specified using any of the following syntaxes:

```
$a = 1.234; $a = 1.2e3; $a = 7E-10;
```

The size of a float is platform-dependent, although a maximum of $\sim 1.8e308$ with a precision of roughly 14 decimal digits is a common value (that's 64 bit IEEE format).

Floating point precision

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 7.999999999...

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 0.3333333... ..

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

A string is series of characters. In PHP, a character is the same as a byte, that is, there are exactly 256 different characters possible. This also implies that PHP has no native support of Unicode.

Note: It is no problem for a string to become very large. There is no practical bound to the size of strings imposed by PHP, so there is no reason at all to worry about long strings.

Syntax

A string literal can be specified in three different ways.

- single quoted
- double quoted
- heredoc syntax

Single quoted

The easiest way to specify a simple string is to enclose it in single quotes (the character `'`).

To specify a literal single quote, you will need to escape it with a backslash (`\`), like in many other languages. If a backslash needs to occur before a single quote or at the end of the string, you need to double it. Note that if you try to escape any other character, the backslash too will be printed! So usually there is no need to escape the backslash itself.

Note: In PHP 3, a warning will be issued at the `E_NOTICE` level when this happens.

Note: Unlike the two other syntaxes, variables will *not* be expanded when they occur in single quoted strings.

```
echo 'this is a simple string';
echo 'You can also have embedded newlines in strings,
like this way.';
echo 'Arnold once said: "I\'ll be back"';
// output: ... "I'll be back"
echo 'Are you sure you want to delete C:\\*.??';
// output: ... delete C:\*.??
echo 'Are you sure you want to delete C:\*.??';
// output: ... delete C:\*.??
echo 'I am trying to include at this point: \n a newline';
// output: ... this point: \n a newline
```

Double quoted

If the string is enclosed in double-quotes (`"`), PHP understands more escape sequences for special characters:

Table 6-1. Escaped characters

sequence	meaning
<code>\n</code>	linefeed (LF or 0x0A (10) in ASCII)
<code>\r</code>	carriage return (CR or 0x0D (13) in ASCII)
<code>\t</code>	horizontal tab (HT or 0x09 (9) in ASCII)
<code>\\</code>	backslash
<code>\\$</code>	dollar sign
<code>\"</code>	double-quote

sequence	meaning
<code>\[0-7]{1,3}</code>	the sequence of characters matching the regular expression is a character in octal notation
<code>\x[0-9A-Fa-f]{1,2}</code>	the sequence of characters matching the regular expression is a character in hexadecimal notation

Again, if you try to escape any other character, the backslash will be printed too!

But the most important pre of double-quoted strings is the fact that variable names will be expanded. See string parsing for details.

Here doc

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. Also, the identifier 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.

Warning

It is very important to note that the line with the closing identifier contains no other characters, except *possibly* a semicolon (;). That means especially that the identifier *may not be indented*, and there may not be any spaces or tabs after or before the semicolon.

Probably the nastiest gotcha is that there may also not be a carriage return (`\r`) at the end of the line, only a form feed, AKA newline (`\n`). Since Microsoft Windows uses the sequence `\r\n` as a line terminator, your heredoc may not work if you write your script in a Windows editor. However, most programming editors provide a way to save your files with a UNIX line terminator.

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 6-2. 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.

Variable parsing

When a string is specified in double quotes or with heredoc, variables are parsed within it.

There are two types of syntax, a simple one and a complex one. The simple syntax is the most common and convenient, it provides a way to parse a variable, an array value, or an object property.

The complex syntax was introduced in PHP 4, and can be recognised by the curly braces surrounding the expression.

Simple syntax

If a dollar sign (\$) is encountered, the parser will greedily take as much tokens as possible to form a valid variable name. Enclose the variable name in curly braces if you want to explicitly specify the end of the name.

```

$beer = 'Heineken';
echo "$beer's taste is great"; // works, "'" is an invalid character for varnames
echo "He drunk some $beers"; // won't work, 's' is a valid character for varnames
echo "He drunk some ${beer}s"; // works

```

Similarly, you can also have an array index or an object property parsed. With array indices, the closing square bracket (]) marks the end of the index. For object properties the same rules apply as to simple variables, though with object properties there doesn't exist a trick like the one with variables.

```
$fruits = array( 'strawberry' => 'red' , 'banana' => 'yellow' );

// note that this works differently outside string-quotes.
echo "A banana is $fruits[banana].";

echo "This square is $square->width meters broad.";

// Won't work. For a solution, see the complex syntax.
echo "This square is $square->width00 centimeters broad.";
```

For anything more complex, you should use the complex syntax.

Complex (curly) syntax

This isn't called complex because the syntax is complex, but because you can include complex expressions this way.

In fact, you can include any value that is in the namespace in strings with this syntax. You simply write the expression the same way as you would outside the string, and then include it in { and }. Since you can't escape '{', this syntax will only be recognised when the \$ is immediately following the {. (Use "\\$" or "\\\$" to get a literal "{\$"). Some examples to make it clear:

```
$great = 'fantastic';
echo "This is { $great}"; // won't work, outputs: This is { fantastic}
echo "This is {$great}"; // works, outputs: This is fantastic
echo "This square is {$square->width}00 centimeters broad.";
echo "This works: {$arr[4][3]}";

// This is wrong for the same reason
// as $foo[bar] is wrong outside a string.
echo "This is wrong: {$arr[foo][3]}";

echo "You should do it this way: {$arr['foo'][3]}";
echo "You can even write {$obj->values[3]->name}";
echo "This is the value of the var named $name: {${$name}}";
```

String access by character

Characters within strings may be accessed by specifying the zero-based offset of the desired character after the string in curly braces.

Note: For backwards compatibility, you can still use the array-braces. However, this syntax is deprecated as of PHP 4.

Example 6-3. 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};
?>
```

Useful functions

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.

There are a lot of useful functions for string modification.

See the string functions section for general functions, the regular expression functions for advanced find&replacing (in two tastes: Perl and POSIX extended).

There are also functions for URL-strings, and functions to encrypt/decrypt strings (mcrypt and mhash).

Finally, if you still didn't find what you're looking for, see also the character type functions.

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 float 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 float (11.5)
$foo = 1 + "-1.3e3";        // $foo is float (-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 float (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

An array in PHP is actually an ordered map. A map is a type that maps *values* to *keys*. This type is optimized in several ways, so you can use it as a real array, or a list (vector), hashtable (which is an implementation of a map), dictionary, collection, stack, queue and probably more. Because you can have another PHP-array as a value, you can also quite easily simulate trees.

Explanation of those structures is beyond the scope of this manual, but you'll find at least one example for each of those structures. For more information about those structures, we refer you to external literature about this broad topic.

Syntax

Specifying with array()

An array can be created by the `array()` language-construct. It takes a certain number of comma-separated `key => value` pairs.

A `key` is either a nonnegative integer or a string. If a key is the standard representation of a non-negative integer, it will be interpreted as such (i.e. `'8'` will be interpreted as 8, while `'08'` will be interpreted as `'08'`).

A `value` can be anything.

If you omit a key, the maximum of the integer-indices is taken, and the new key will be that maximum + 1. If no integer-indices exist yet, the key will be 0 (zero). If you specify a key that already has a value assigned to it, that value will be overwritten.

```
array( [key =>] value
      , ...
      )
// key is either string or nonnegative integer
// value can be anything
```

Creating/modifying with square-bracket syntax

You can also modify an existing array, by explicitly setting values.

This is done by assigning values to the array while specifying the key in brackets. You can also omit the key, add an empty pair of brackets (`[]`) to the variable-name in that case.

```
$arr[key] = value;
$arr[] = value;
// key is either string or nonnegative integer
// value can be anything
```

If `$arr` doesn't exist yet, it will be created. So this is also an alternative way to specify an array. To change a certain value, just assign a new value to it. If you want to remove a key/value pair, you need to `unset()` it.

Useful functions

There are quite some useful function for working with arrays, see the array-functions section.

Note: The `unset()` function allows unsetting keys of an array. Be aware that the array will NOT be reindexed.

```
$a = array( 1 => 'one', 2 => 'two', 3 => 'three' );
unset( $a[2] );
/* will produce an array that would have been defined as
   $a = array( 1=>'one', 3=>'three' );
   and NOT
   $a = array( 1 => 'one', 2 => 'three' );
*/
```

The `foreach` control structure exists specifically for arrays. It provides an easy way to traverse an array.

Array do's and don'ts

Why is `$foo[bar]` wrong?

You might have seen the following syntax in old scripts:

```
$foo[bar] = 'enemy';
echo $foo[bar];
// etc
```

This is wrong, but it works. Then, why is it wrong? The reason is that, as stated in the syntax section, there must be an expression between the square brackets ('[' and ']'). That means that you can write things like this:

```
echo $arr[ foo(true) ];
```

This is an example of using a function return value as the array index. PHP knows also about constants, and you may have seen the `E_*` before.

```
$error_descriptions[E_ERROR] = "A fatal error has occurred";
$error_descriptions[E_WARNING] = "PHP issued a warning";
$error_descriptions[E_NOTICE] = "This is just an informal notice";
```

Note that `E_ERROR` is also a valid identifier, just like `bar` in the first example. But the last example is in fact the same as writing:

```
$error_descriptions[1] = "A fatal error has occurred";
$error_descriptions[2] = "PHP issued a warning";
$error_descriptions[8] = "This is just an informal notice";
```

because `E_ERROR` equals 1, etc.

Then, how is it possible that `$foo[bar]` works? It works, because `bar` is due to its syntax expected to be a constant expression. However, in this case no constant with the name `bar` exists. PHP now assumes that you meant `bar` literally, as the string `"bar"`, but that you forgot to write the quotes.

So why is it bad then?

At some point in the future, the PHP team might want to add another constant or keyword, and then you get in trouble. For example, you already cannot use the words `empty` and `default` this way, since they are special keywords.

And, if these arguments don't help: this syntax is simply deprecated, and it might stop working some day.

Note: When you turn `error_reporting` to `E_ALL`, you will see that PHP generates warnings whenever this construct is used. This is also valid for other deprecated 'features'. (put the line `error_reporting(E_ALL);` in your script)

Note: Inside a double-quoted string, an other syntax is valid. See variable parsing in strings for more details.

Examples

The array type in PHP is very versatile, so here will be some examples to show you the full power of arrays.

```
// this
$a = array( 'color' => 'red'
           , 'taste' => 'sweet'
           , 'shape' => 'round'
           , 'name'  => 'apple'
           ,
           , 4 // key will be 0
           );

// is completely equivalent with
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name'] = 'apple';
$a[] = 4; // key will be 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// will result in the array array( 0 => 'a' , 1 => 'b' , 2 => 'c' ),
// or simply array('a', 'b', 'c')
```

Example 6-4. Using array()

```
// Array as (property-)map
$map = array( 'version' => 4
            , 'OS'      => 'Linux'
            , 'lang'    => 'english'
            , 'short_tags' => true
            );

// strictly numerical keys
$array = array( 7
              , 8
              , 0
              , 156
              , -10
              );
// this is the same as array( 0 => 7, 1 => 8, ...)

$switching = array( 10 // key = 0
                  , 5   => 6
```



```

, 3    => 7
, 'a'  => 4
,      11 // key = 6 (maximum of integer-indices was 5)
, '8'  => 2 // key = 8 (integer!)
, '02' => 77 // key = '02'
, 0    => 12 // the value 10 will be overwritten by 12
);

// empty array
$empty = array();

```

Example 6-5. Collection

```

$colors = array('red', 'blue', 'green', 'yellow');

foreach ( $colors as $color ) {
    echo "Do you like $color?\n";
}

/* output:
Do you like red?
Do you like blue?
Do you like green?
Do you like yellow?
*/

```

Note that it is currently not possible to change the values of the array directly in such a loop. A workaround is the following:

Example 6-6. Collection

```

foreach ($colors as $key => $color) {
    // won't work:
    //$color = strtoupper($color);

    //works:
    $colors[$key] = strtoupper($color);
}
print_r($colors);

/* output:
Array
(
    [0] => RED
    [1] => BLUE

```

```

    [2] => GREEN
    [3] => YELLOW
)
*/

```

This example creates a one-based array.

Example 6-7. One-based index

```

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

/* output:
Array
(
    [1] => 'January'
    [2] => 'February'
    [3] => 'March'
)
*/

```

Example 6-8. Filling real array

```

// fill an array with all items from a directory
$handle = opendir('.');
while ($file = readdir($handle))
{
    $files[] = $file;
}
closedir($handle);

```

Arrays are ordered. You can also change the order using various sorting-functions. See array-functions for more information.

Example 6-9. Sorting array

```

sort($files);
print_r($files);

```

Because the value of an array can be everything, it can also be another array. This way you can make recursive and multi-dimensional arrays.

Example 6-10. Recursive and multi-dimensional arrays

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

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](#).

Resource

A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions. See the [appendix](#) for a listing of all these functions and the corresponding resource types.

Note: The resource type was introduced in PHP 4

Freeing resources

Due to the reference-counting system introduced with PHP4's Zend-engine, it is automatically detected when a resource is no longer referred to (just like Java). When this is the case, all resources that were in use for this resource are made free by the garbage collector. For this reason, it is rarely ever necessary to free the memory manually by using some `free_result` function.

Note: Persistent database-links are special, they are *not* destroyed by the gc. See also [persistent links](#)

NULL

The special `NULL` value represents that a variable has no value. `NULL` is the only possible value of type `NULL`.

Note: The null type was introduced in PHP 4

Syntax

There is only one value of type `NULL`, and that is the case-insensitive keyword `NULL`.

```
$var = NULL;
```

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 float, then all operands are evaluated as floats, and the result will be a float. 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 += 2; // $foo is now an integer (2)
$foo = $foo + 1.3; // $foo is now a float (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 use the `var_dump()` function.

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 = (float) $foo;    // $bar is a float
```

The casts allowed are:

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

Note: Instead of casting a variable to string, you can also enclose the variable in double quotes.

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 more info, see these sections:

- [Converting to boolean](#)
- [Converting to integer](#)

When casting or forcing a conversion from array to string, the result will be the word `Array`. When casting or forcing a conversion from object to string, the result will be the word `Object`.

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
$täyte = '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 $bar;
echo $foo;                   // $foo is altered too.
?>
```

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. For a listing of these variables, please see the section Predefined variables.

Warning

In PHP 4.2.0 and later, the default set of predefined variables which are available in the global scope has changed. Individual input and server variables are *by default* no longer placed directly into the global scope; rather, they are placed into the following superglobal arrays.

You can still force the old behaviour by setting `register_globals` to 'On' in your `php.ini` file.

For more information and background on this change, please see the PHP 4.1.0 Release Announcement (http://www.php.net/release_4_1_0.php).

From version 4.1.0 onward, PHP provides a set of predefined arrays containing variables from the web server (if applicable), the environment, and user input. These new arrays are rather special in that they are automatically global--i.e., automatically available in every scope. For this reason, they are often known as 'autoglobals' or 'superglobals'. (There is no mechanism in PHP for user-defined superglobals.) The superglobals are listed below; however, for a listing of their contents and further discussion on PHP predefined variables and their natures, please see the section Predefined variables.

PHP Superglobals

`$GLOBALS`

Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.

`$_SERVER`

Variables set by the web server or otherwise directly related to the execution environment of the current script. Analogous to the old `$HTTP_SERVER_VARS` array (which is still available, but deprecated).

`$_GET`

Variables provided to the script via HTTP GET. Analogous to the old `$HTTP_GET_VARS` array (which is still available, but deprecated).

`$_POST`

Variables provided to the script via HTTP POST. Analogous to the old `$HTTP_POST_VARS` array (which is still available, but deprecated).

`$_COOKIE`

Variables provided to the script via HTTP cookies. Analogous to the old `$HTTP_COOKIE_VARS` array (which is still available, but deprecated).

`$_FILES`

Variables provided to the script via HTTP post file uploads. Analogous to the old `$HTTP_POST_FILES` array (which is still available, but deprecated). See POST method uploads for more information.

`$_ENV`

Variables provided to the script via the environment. Analogous to the old `$HTTP_ENV_VARS` array (which is still available, but deprecated).

`$_REQUEST`

Variables provided to the script via any user input mechanism, and which therefore cannot be trusted. Note: when running on the command line, this will *not* include the `argv` and `argc` entries; these are present in the `$_SERVER` array. The presence and order of variable inclusion in this array is defined according to the `variables_order` configuration directive. This array has no direct analogue in versions of PHP prior to 4.1.0.

`$_SESSION`

Variables which are currently registered to a script's session. Analogous to the old `$HTTP_SESSION_VARS` array (which is still available, but deprecated). See the Session handling functions section for more information.

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.

Note: The `magic_quotes_gpc` configuration directive affects Get, Post and Cookie values. If turned on, value (It's "PHP!") will automatically become (It\'s \"PHP!\"). Escaping is needed for DB insertion. Also see `addslashes()`, `stripslashes()` and `magic_quotes_sybase`.

PHP also understands arrays in the context of form variables (see the related [faq](#)). 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_array()`, `is_float()`, `is_int()`, `is_object()`, and `is_string()`.

Chapter 8. Constants

A constant is a identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script. (The 'magic constants' `__FILE__` and `__LINE__` appear to be an exception to this, but they're not actually constants.) A constant is case-sensitive by default. By convention constant identifiers are always uppercase.

The name of a constant follows the same rules as any label in PHP. A valid constant 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).

The scope of a constant is global--you can access it anywhere in your script without regard to scope.

Syntax

You can define a constant by using the `define()`-function. Once a constant is defined, it can never be changed or undefined.

Only scalar data (boolean, integer, double and string) can be contained in constants.

You can get the value of a constant by simply specifying its name. Unlike with variables, you should *not* prepend a constant with a `$`. You can also use the function `constant()`, to read a constant's value, if you are to obtain the constant's name dynamically. Use `get_defined_constants()` to get a list of all defined constants.

Note: Constants and (global) variables are in a different namespace. This implies that for example `TRUE` and `$TRUE` are generally different.

If you use an undefined constant, PHP assumes that you mean the name of the constant itself. A notice will be issued when this happens. Use the `defined()`-function if you want to know if a constant is set.

These are the differences between constants and variables:

- Constants do not have a dollar sign (`$`) before them;
- Constants may only be defined using the `define()` function, not by simple assignment;
- Constants may be defined and 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.

Example 8-1. Defining Constants

```
<?php
```

```
define("CONSTANT", "Hello world.");  
echo CONSTANT; // outputs "Hello world."  
echo Constant; // outputs "Constant" and issues a notice.  
?>
```

Predefined constants

PHP provides a large number of predefined constants to any script which it runs. Many of these constants, however, are created by various extensions, and will only be present when those extensions are available, either via dynamic loading or because they have been compiled in.

A list of predefined constants is available in the section [Predefined constants](#).

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 the second subexpression is evaluated, and that is the result of the conditional expression. Otherwise, the third subexpression is evaluated, and that is the value.

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

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

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

$f = double($d++);   /* assign twice the value of $d <emphasis>before</emphasis>
```



```

                                the increment, 2*6 = 12 to $f */
$g = double(++$e); /* assign twice the value of $e <emphasis>after</emphasis>
                                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`. The constants `TRUE` and `FALSE` (case-insensitive) are the two possible boolean values. When necessary, an expression is automatically converted to boolean. See the section about type-casting for details about how.

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

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 10-1. Operator Precedence

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

Arithmetic Operators

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

Table 10-2. Arithmetic Operators

Example	Name	Result
$\$a + \b	Addition	Sum of $\$a$ and $\$b$.

Example	Name	Result
$\$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 a float value anytime, even if the two operands are integers (or strings that get converted to integers).

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. If both the left- and right-hand parameters are strings, the bitwise operator will operate on the characters in this string.

```
<?php
    echo 12 ^ 9; // Outputs '5'

    echo "12" ^ "9"; // Outputs the Backspace character (ascii 8)
                    // ('1' (ascii 49)) ^ ('9' (ascii 57)) = #8

    echo "hallo" ^ "hello"; // Outputs the ascii values #0 #4 #0 #0 #0
                            // 'a' ^ 'e' = #4
?>
```

Table 10-3. Bitwise Operators

Example	Name	Result
$\$a \& \b	And	Bits that are set in both $\$a$ and $\$b$ are set.
$\$a \b	Or	Bits that are set in either $\$a$ or $\$b$ are set.
$\$a \wedge \b	Xor	Bits that are set in $\$a$ or $\$b$ but not both are set.
$\sim \$a$	Not	Bits that are set in $\$a$ are not set, and vice versa.
$\$a \ll \b	Shift left	Shift the bits of $\$a$ $\$b$ steps to the left (each step means "multiply by two")
$\$a \gg \b	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 10-4. Comparison Operators

Example	Name	Result
---------	------	--------

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 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 file error */
$my_file = @file ('non_existent_file') or
    die ("Failed opening file: error was '$php_errormsg'");

// this works for any expression, not just functions:
```

```
$value = @$cache[$key];
// will not issue a notice if the index $key doesn't exist.

?>
```

Note: The @-operator works only on expressions. A simple rule of thumb is: if you can take the value of something, you can prepend the @ operator to it. For instance, you can prepend it to variables, function and include() calls, constants, and so forth. You cannot prepend it to function or class definitions, or conditional structures such as `if` and `foreach`, and so forth.

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 or `shell_exec()` is disabled.

See also `escapeshellcmd()`, `exec()`, `passthru()`, `popen()`, `shell_exec()`, and `system()`.

Incrementing/Decrementing Operators

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

Table 10-5. Increment/decrement Operators

Example	Name	Effect
++\$a	Pre-increment	Increments \$a by one, then returns \$a.
\$a++	Post-increment	Returns \$a, then increments \$a by one.
--\$a	Pre-decrement	Decrements \$a by one, then returns \$a.
\$a--	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 10-6. 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.)

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 Boolean value. If `expr` evaluates to `TRUE`, PHP will execute `statement`, and if it evaluates to `FALSE` - it'll ignore it. More information about what values evaluate to `FALSE` can be found in the 'Converting to boolean' section>.

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 as with the `each()` construct and changes to the array element returned are not reflected in the original array.

Note: `foreach` does not support the ability to suppress error messages using '@'.

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] => $v.\n";
    $i++;
}
```

```

/* 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";
}

/* foreach example 4: multi-dimensional arrays */

$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";

foreach($a as $v1) {
    foreach ($v1 as $v2) {
        print "$v2\n";
    }
}

/* foreach example 5: dynamic arrays */

foreach(array(1, 2, 3, 4, 5) as $v) {
    print "$v\n";
}

```

break

`break` ends execution of the current `for`, `foreach` `while`, `do..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 "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Middle<br>\n";
        while (1) {
            echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;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, and should be the last `case` statement. 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;

```

declare

The `declare` construct is used to set execution directives for a block of code. The syntax of `declare` is similar to the syntax of other flow control constructs:

```
declare (directive) statement
```

The `directive` section allows the behavior of the `declare` block to be set. Currently only one directive is recognized: the `ticks` directive. (See below for more information on the `ticks` directive)

The statement part of the `declare` block will be executed - how it is executed and what side-effects occur during execution may depend on the directive set in the `directive` block.

Ticks

A tick is an event that occurs for every N low-level statements executed by the parser within the `declare` block. The value for N is specified using `ticks=N` within the `declare` blocks's `directive` section.

The event(s) that occurs on each tick is specified using the `register_tick_function()`. See the example below for more details. Note that more than one event can occur for each tick.

Example 11-1. Profile a section of PHP code

```
<?php
// A function that records the time when it is called
function profile ($dump = FALSE)
{
    static $profile;

    // Return the times stored in profile, then erase it
    if ($dump) {
        $temp = $profile;
        unset ($profile);
        return ($temp);
    }

    $profile[] = microtime ();
}

// Set up a tick handler
register_tick_function("profile");

// Initialize the function before the declare block
profile ();

// Run a block of code, throw a tick every 2nd statement
declare (ticks=2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text (md5($x), md5($x*$x)), "<br />";
    }
}

// Display the data stored in the profiler
print_r (profile (TRUE));
?>
```

The example profiles the PHP code within the 'declare' block, recording the time at which every second low-level statement in the block was executed. This information can then be used to find the slow areas

within particular segments of code. This process can be performed using other methods: using ticks is more convenient and easier to implement.

Ticks are well suited for debugging, implementing simple multitasking, backgrounded I/O and many other tasks.

See also `register_tick_function()` and `unregister_tick_function()`.

return

If called from within a function, the `return()` statement immediately ends execution of the current function, and returns its argument as the value of the function call. `return()` will also end the execution of an `eval()` statement or script file.

If called from the global scope, then execution of the current script file is ended. If the current script file was `include()`d or `require()`d, then control is passed back to the calling file. Furthermore, if the current script file was `include()`d, then the value given to `return()` will be returned as the value of the `include()` call. If `return()` is called from within the main script file, then script execution ends. If the current script file was named by the `auto_prepend_file` or `auto_append_file` configuration options in the configuration file, then that script file's execution is ended.

For more information, see [Returning values](#).

Note: Note that since `return()` is a language construct and not a function, the parentheses surrounding its arguments are *not* required--in fact, it is more common to leave them out than to use them, although it doesn't matter one way or the other.

require()

The `require()` statement includes and evaluates the specific file.

`require()` includes and evaluates a specific file. Detailed information on how this inclusion works is described in the documentation for `include()`.

`require()` and `include()` are identical in every way except how they handle failure. `include()` produces a Warning while `require()` results in a Fatal Error. In other words, don't hesitate to use `require()` if you want a missing file to halt processing of the page. `include()` does not behave this way, the script will continue regardless. Be sure to have an appropriate `include_path` setting as well.

Example 11-2. Basic `require()` examples

```
<?php
require 'prepend.php' ;
```

```
require $somefile;

require ('somefile.txt');

?>
```

See the `include()` documentation for more examples.

Note: Prior to PHP 4.0.2, the following applies: `require()` will always attempt to read the target file, even if the line it's on never executes. The conditional statement won't affect `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.

See also `include()`, `require_once()`, `include_once()`, `eval()`, `file()`, `readfile()`, `virtual()` and `include_path`.

include()

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

The documentation below also applies to `require()`. The two constructs are identical in every way except how they handle failure. `include()` produces a Warning while `require()` results in a Fatal Error. In other words, use `require()` if you want a missing file to halt processing of the page. `include()` does not behave this way, the script will continue regardless. Be sure to have an appropriate `include_path` setting as well.

When a file is included, 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, from that point forward.

Example 11-3. Basic `include()` example

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>

test.php
<?php

echo "A $color $fruit"; // A
```

```
include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

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. So, it will follow the variable scope of that function.

Example 11-4. Including within functions

```
<?php

function foo()
{
    global $color;

        include 'vars.php';

        echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so      *
 * $fruit is NOT available outside of this *
 * scope. $color is because we declared it *
 * as global.                               */

foo();                                     // A green apple
echo "A $color $fruit";                   // A green

?>
```

When a file is included, 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.

If "URL fopen wrappers" are enabled in PHP (which they are in the default configuration), you can specify the file to be included using an URL (via HTTP) instead of a local pathname. If the target server interprets the target file as PHP code, variables may be passed to the included file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as including 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.

Example 11-5. include() through HTTP

```

<?php

/* This example assumes that www.example.com is configured to parse .php *
 * files and not .txt files. Also, 'Works' here means that the variables *
 * $foo and $bar are available within the included file.                    */

// Won't work; file.txt wasn't handled by www.example.com as PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Won't work; looks for a file named 'file.php?foo=1&bar=2' on the
// local filesystem.
include 'file.php?foo=1&bar=2';

// Works.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt'; // Works.
include 'file.php'; // Works.

?>

```

See also Remote files, fopen() and file() for related information.

Because include() and require() are special language constructs, you must enclose them within a statement block if it's inside a conditional block.

Example 11-6. include() and conditional blocks

```

<?php

// 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;
}

?>

```

Handling Returns: It is possible to execute a `return()` statement inside an included file in order to terminate processing in that file and return to the script which called it. Also, it's possible to return values from included files. You can take the value of the include call as you would a normal function.

Note: 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.

Example 11-7. `include()` and the `return()` statement

```
return.php
<?php

$var = 'PHP';

return $var;

?>

noreturn.php
<?php

$var = 'PHP';

?>

testreturns.php
<?php

$foo = include 'return.php';

echo $foo; // prints 'PHP'

$bar = include 'noreturn.php';

echo $bar; // prints 1

?>
```

`$bar` is the value 1 because the include was successful. Notice the difference between the above examples. The first uses `return()` within the included file while the other does not. A few other ways to

"include" files into variables are with `fopen()`, `file()` or by using `include()` along with Output Control Functions.

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

require_once()

The `require_once()` statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the `require()` statement, with the only difference being that if the code from a file has already been included, it will not be included again. See the documentation for `require()` for more information on how this statement works.

`require_once()` should be used in cases where 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 reassignments, etc.

For examples on using `require_once()` and `include_once()`, look at the PEAR (<http://pear.php.net/>) code included in the latest PHP source code distributions.

Note: `require_once()` was added in PHP 4.0.1pl2

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 only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once.

`include_once()` should be used in cases where 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 reassignments, etc.

For more examples on using `require_once()` and `include_once()`, look at the PEAR (<http://pear.php.net/>) code included in the latest PHP source code distributions.

Note: `include_once()` was added in PHP 4.0.1pl2

See also `include()`, `require()`, `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.'
```

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. This causes the function to end its execution immediately and pass control back to the line from which it was called. See `return()` for more information.

```
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();
```

For more information on references, please check out [References Explained](#).

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.

Variable functions won't work with language constructs such as `echo()`, `unset()`, `isset()`, `empty()` and `include()`. Although, the construct `print()` is an exception and will work. This is one of the major differences between PHP functions and language constructs.

Example 12-1. Variable function example

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

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

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

See also `variable variables` and `function_exists()`.

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.

Caution

The following cautionary notes are valid for PHP 4.

The name `stdClass` is used internally by Zend and is reserved. You cannot have a class named `stdClass` in PHP.

The function names `__sleep` and `__wakeup` are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them. See below for more information.

PHP reserves all function names starting with `__` as magical. It is recommended that you do not use function names with `__` in PHP unless you want some documented magic functionality.

Note: In PHP 4, only constant initializers for `var` variables are allowed. To initialize variables with non-constant values, you need an initialization function which is called automatically when an object is being constructed from the class. Such a function is called a constructor (see below).

```
<?php
/* None of these will work in PHP 4. */
class Cart
{
    var $todays_date = date("Y-m-d");
    var $name = $firstname;
    var $owner = 'Fred ' . 'Jones';
    var $items = array("VCR", "TV");
}

/* This is how it should be done. */
class Cart
{
    var $todays_date;
    var $name;
    var $owner;
    var $items;

    function Cart()
    {
        $this->todays_date = date("Y-m-d");
        $this->name = $GLOBALS['firstname'];
        /* etc. . . */
    }
}
```

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.

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

$another_cart = new Cart;
$another_cart->add_item("0815", 3);
```

This creates the objects `$cart` and `$another_cart`, both of the class `Cart`. The function `add_item()` of the `$cart` object is being called to add 1 item of article number 10 to the `$cart`. 3 items of article number 0815 are being added to `$another_cart`.

Both, `$cart` and `$another_cart`, have functions `add_item()`, `remove_item()` and a variable `items`. These are distinct functions and variables. You can think of the objects as something similar to directories in a filesystem. In a filesystem you can have two different files `README.TXT`, as long as they are in

different directories. Just like with directories where you'll have to type the full pathname in order to reach each file from the toplevel directory, you have to specify the complete name of the function you want to call: In PHP terms, the toplevel directory would be the global namespace, and the pathname separator would be `->`. Thus, the names `$cart->items` and `$another_cart->items` name two different variables. Note that the variable is named `$cart->items`, not `$cart->$items`, that is, a variable name in PHP has only a single dollar sign.

```
// correct, single $
$cart->items = array("10" => 1);

// invalid, because $cart->$items becomes $cart->""
$cart->$items = array("10" => 1);

// correct, but may or may not be what was intended:
// $cart->$myvar becomes $cart->items
$myvar = 'items';
$cart->$myvar = array("10" => 1);
```

Within a class definition, you do not know under which name the object will be accessible in your program: at the time the `Cart` class was written, it was unknown that the object will be named `$cart` or `$another_cart` later. Thus, you cannot write `$cart->items` within the `Cart` class itself. Instead, in order to be able to access it's own functions and variables from within a class, one can use the pseudo-variable `$this` which can be read as 'my own' or 'current object'. Thus, `'$this->items[$artnr] += $num'` can be read as 'add `$num` to the `$artnr` counter of my own items array' or 'add `$num` to the `$artnr` counter of the items array within the current object'.

Note: There are some nice functions to handle classes and objects. You might want to take a look at the [Class/Object Functions](#)

extends

Often you need classes with similar variables and functions to another existing class. In fact, it is good practice to define a generic class which can be used in all your projects and adapt this class for the needs of each of your specific projects. To facilitate this, classes can be extensions of other classes. The extended or derived class has all variables and functions of the base class (this is called 'inheritance' despite the fact that nobody died) and what you add in the extended definition. It is not possible to subtract from a class, that is, to undefine any existing functions or variables. An extended class is always dependent on a single base class, that is, multiple inheritance is not supported. Classes are extended using the keyword 'extends'.

```
class Named_Cart extends Cart
{
```



```

var $owner;

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

```

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

```

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

```

Constructors

Caution

In PHP 3 and PHP 4 constructors behave differently. The PHP 4 semantics are strongly preferred.

Constructors are functions in a class that are automatically called when you create a new instance of a class with `new`. In PHP 3, a function becomes a constructor when it has the same name as the class. In PHP 4, a function becomes a constructor, when it has the same name as the class it is defined in - the difference is subtle, but crucial (see below).

```

// Works in PHP 3 and PHP 4.
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 take

arguments and these arguments can be optional, which makes them much more useful. To be able to still use the class without parameters, all parameters to constructors should be made optional by providing default values.

```
// Works in PHP 3 and PHP 4.
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

In PHP 3, derived classes and constructors have a number of limitations. The following examples should be read carefully to understand these limitations.

```
class A
{
    function A()
    {
        echo "I am the constructor of A.<br>\n";
    }
}

class B extends A
{
    function C()
    {
        echo "I am a regular function.<br>\n";
    }
}

// no constructor is being called in PHP 3.
$b = new B;
```

In PHP 3, no constructor is being called in the above example. The rule in PHP 3 is: 'A constructor is a function of the same name as the class.'. The name of the class is B, and there is no function called B() in class B. Nothing happens.

This is fixed in PHP 4 by introducing another rule: If a class has no constructor, the constructor of the base class is being called, if it exists. The above example would have printed 'I am the constructor of A.
' in PHP 4.

```
class A
{
    function A()
    {
        echo "I am the constructor of A.<br>\n";
    }

    function B()
    {
        echo "I am a regular function named B in class A.<br>\n";
        echo "I am not a constructor in A.<br>\n";
    }
}

class B extends A
{
    function C()
    {
        echo "I am a regular function.<br>\n";
    }
}

// This will call B() as a constructor.
$b = new B;
```

In PHP 3, the function B() in class A will suddenly become a constructor in class B, although it was never intended to be. The rule in PHP 3 is: 'A constructor is a function of the same name as the class.'. PHP 3 does not care if the function is being defined in class B, or if it has been inherited.

This is fixed in PHP 4 by modifying the rule to: 'A constructor is a function of the same name as the class it is being defined in.'. Thus in PHP 4, the class B would have no constructor function of its own and the constructor of the base class would have been called, printing 'I am the constructor of A.
'.

Caution

Neither PHP 3 nor PHP 4 call constructors of the base class automatically from a constructor of a derived class. It is your responsibility to propagate the call to constructors upstream where appropriate.

Note: There are no destructors in PHP 3 or PHP 4. You may use `register_shutdown_function()` instead to simulate most effects of destructors.

Destructors are functions that are called automatically when an object is destroyed, either with `unset()` or by simply going out of scope. There are no destructors in PHP.

::

Caution

The following is valid for PHP 4 only.

Sometimes it is useful to refer to functions and variables in base classes or to refer to functions in classes that have not yet any instances. The `::` operator is being used for this.

```
class A
{
    function example()
    {
        echo "I am the original function A::example().<br>\n";
    }
}

class B extends A
{
    function example()
    {
        echo "I am the redefined function B::example().<br>\n";
        A::example();
    }
}

// there is no object of class A.
// this will print
//   I am the original function A::example().<br>
A::example();

// create an object of class B.
$b = new B;

// this will print
//   I am the redefined function B::example().<br>
//   I am the original function A::example().<br>
$b->example();
```

The above example calls the function `example()` in class A, but there is no object of class A, so that we cannot write `$a->example()` or similar. Instead we call `example()` as a 'class function', that is, as a function of the class itself, not any object of that class.

There are class functions, but there are no class variables. In fact, there is no object at all at the time of the call. Thus, a class function may not use any object variables (but it can use local and global variables), and it may not use `$this` at all.

In the above example, class B redefines the function `example()`. The original definition in class A is shadowed and no longer available, unless you are referring specifically to the implementation of `example()` in class A using the `::`-operator. Write `A::example()` to do this (in fact, you should be writing `parent::example()`, as shown in the next section).

In this context, there is a current object and it may have object variables. Thus, when used from WITHIN an object function, you may use `$this` and object variables.

parent

You may find yourself writing code that refers to variables and functions in base classes. This is particularly true if your derived class is a refinement or specialisation of code in your base class.

Instead of using the literal name of the base class in your code, you should be using the special name `parent`, which refers to the name of your base class as given in the `extends` declaration of your class. By doing this, you avoid using the name of your base class in more than one place. Should your inheritance tree change during implementation, the change is easily made by simply changing the `extends` declaration of your class.

```
class A
{
    function example()
    {
        echo "I am A::example() and provide basic functionality.<br>\n";
    }
}

class B extends A
{
    function example()
    {
        echo "I am B::example() and provide additional functionality.<br>\n";
        parent::example();
    }
}

$b = new B;

// This will call B::example(), which will in turn call A::example().
$b->example();
```

Serializing objects - objects in sessions

Note: In PHP 3, objects will lose their class association throughout the process of serialization and unserialization. The resulting variable is of type object, but has no class and no methods, thus it is pretty useless (it has become just like an array with a funny syntax).

Caution

The following information is valid for PHP 4 only.

`serialize()` returns a string containing a byte-stream representation of any value that can be stored in PHP. `unserialize()` can use this string to recreate the original variable values. Using `serialize` to save an object will save all variables in an object. The functions in an object will not be saved, only the name of the class.

In order to be able to `unserialize()` an object, the class of that object needs to be defined. That is, if you have an object `$a` of class `A` on `page1.php` and `serialize` this, you'll get a string that refers to class `A` and contains all values of variables contained in `$a`. If you want to be able to `unserialize` this on `page2.php`, recreating `$a` of class `A`, the definition of class `A` must be present in `page2.php`. This can be done for example by storing the class definition of class `A` in an include file and including this file in both `page1.php` and `page2.php`.

```

classa.inc:
    class A
    {
        var $one = 1;

        function show_one()
        {
            echo $this->one;
        }
    }

page1.php:
    include("classa.inc");

    $a = new A;
    $s = serialize($a);
    // store $s somewhere where page2.php can find it.
    $fp = fopen("store", "w");
    fputs($fp, $s);
    fclose($fp);

page2.php:
    // this is needed for the unserialize to work properly.
    include("classa.inc");

    $s = implode("", @file("store"));

```

```

$a = unserialize($s);

// now use the function show_one() of the $a object.
$a->show_one();

```

If you are using sessions and use `session_register()` to register objects, these objects are serialized automatically at the end of each PHP page, and are unserialized automatically on each of the following pages. This basically means that these objects can show up on any of your pages once they become part of your session.

It is strongly recommended that you include the class definitions of all such registered objects on all of your pages, even if you do not actually use these classes on all of your pages. If you don't and an object is being unserialized without its class definition being present, it will lose its class association and become an object of class `stdClass` without any functions available at all, that is, it will become quite useless.

So if in the example above `$a` became part of a session by running `session_register("a")`, you should include the file `classa.inc` on all of your pages, not only `page1.php` and `page2.php`.

The magic functions `__sleep` and `__wakeup`

`serialize()` checks if your class has a function with the magic name `__sleep`. If so, that function is being run prior to any serialization. It can clean up the object and is supposed to return an array with the names of all variables of that object that should be serialized.

The intended use of `__sleep` is to close any database connections that object may have, committing pending data or perform similar cleanup tasks. Also, the function is useful if you have very large objects which need not be saved completely.

Conversely, `unserialize()` checks for the presence of a function with the magic name `__wakeup`. If present, this function can reconstruct any resources that object may have.

The intended use of `__wakeup` is to reestablish any database connections that may have been lost during serialization and perform other reinitialization tasks.

References inside the constructor

Creating references within the constructor can lead to confusing results. This tutorial-like section helps you to avoid problems.

```

class Foo
{
    function Foo($name)
    {
        // create a reference inside the global array $globalref
        global $globalref;
        $globalref[] = &$this;
    }
}

```

```

        // set name to passed value
        $this->setName($name);
        // and put it out
        $this->echoName();
    }

    function echoName()
    {
        echo "<br>", $this->name;
    }

    function setName($name)
    {
        $this->name = $name;
    }
}

```

Let us check out if there is a difference between `$bar1` which has been created using the copy = operator and `$bar2` which has been created using the reference =& operator...

```

$bar1 = new Foo('set in constructor');
$bar1->echoName();
$globalref[0]->echoName();

/* output:
set in constructor
set in constructor
set in constructor */

$bar2 =& new Foo('set in constructor');
$bar2->echoName();
$globalref[1]->echoName();

/* output:
set in constructor
set in constructor
set in constructor */

```

Apparently there is no difference, but in fact there is a very significant one: `$bar1` and `$globalref[0]` are NOT referenced, they are NOT the same variable. This is because "new" does not return a reference by default, instead it returns a copy.

Note: There is no performance loss (since PHP 4 and up use reference counting) returning copies instead of references. On the contrary it is most often better to simply work with copies instead of

references, because creating references takes some time where creating copies virtually takes no time (unless none of them is a large array or object and one of them gets changed and the other(s) one(s) subsequently, then it would be wise to use references to change them all concurrently).

To prove what is written above let us watch the code below.

```
// now we will change the name. what do you expect?
// you could expect that both $bar1 and $globalref[0] change their names...
$bar1->setName('set from outside');

// as mentioned before this is not the case.
$bar1->echoName();
$globalref[0]->echoName();

/* output:
set from outside
set in constructor */

// let us see what is different with $bar2 and $globalref[1]
$bar2->setName('set from outside');

// luckily they are not only equal, they are the same variable
// thus $bar2->name and $globalref[1]->name are the same too
$bar2->echoName();
$globalref[1]->echoName();

/* output:
set from outside
set from outside */
```

Another final example, try to understand it.

```
class A
{
    function A($i)
    {
        $this->value = $i;
        // try to figure out why we do not need a reference here
        $this->b = new B($this);
    }

    function createRef()
    {
        $this->c = new B($this);
    }

    function echoValue()
```

```

    {
        echo "<br>", "class ", get_class($this), ': ', $this->value;
    }
}

class B
{
    function B(&$a)
    {
        $this->a = &$a;
    }

    function echoValue()
    {
        echo "<br>", "class ", get_class($this), ': ', $this->a->value;
    }
}

// try to understand why using a simple copy here would yield
// in an undesired result in the *-marked line
$a =& new A(10);
$a->createRef();

$a->echoValue();
$a->b->echoValue();
$a->c->echoValue();

$a->value = 11;

$a->echoValue();
$a->b->echoValue(); // *
$a->c->echoValue();

/*
output:
class A: 10
class B: 10
class B: 10
class A: 11
class B: 11
class B: 11
*/

```

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: Not using the `&` operator causes a copy of the object to be made. If you use `$this` in the class it will operate on the current instance of the class. The assignment without `&` will copy the instance (i.e. the object) and `$this` will operate on the copy, which is not always what is desired. Usually you want to have a single instance to work with, due to performance and memory consumption issues.

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 = "<errorentry>\n";
    $err .= "\t<datetime>". $dt. "</datetime>\n";
    $err .= "\t<errornum>". $errno. "</errornum>\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 .= "</errorentry>\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@example.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);
}

$sold_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 "<p>Hello $PHP_AUTH_USER.</p>";
    echo "<p>You entered $PHP_AUTH_PW as your password.</p>";
  }
?>
```

Note: Please be careful when coding the HTTP header lines. In order to guarantee maximum compatibility with all clients, the keyword "Basic" should be written with an uppercase "B", the realm string must be enclosed in double (not single) quotes, and exactly one space should precede the "401" code in the "HTTP/1.0 401" header line.

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.

Configuration Note: PHP uses the presence of an `AuthType` directive to determine whether external authentication is in effect. Remember to avoid this directive for the context where you want to use PHP authentication (otherwise each authentication attempt will fail).

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 "<p>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></p>\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. You can use the output buffering functions to delay the script output until you have decided whether or not to set any cookies or send any headers.

Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data, depending on the `register_globals` and `variables_order` configuration variables. If you wish to assign multiple values to a single cookie, just add `[]` to the cookie name.

In PHP 4.1.0 and later, the `$_COOKIE` auto-global array will always be set with any cookies sent from the client. `$HTTP_COOKIE_VARS` is also set in earlier versions of PHP when the `track_vars` configuration variable is set.

For more details, including notes on browser bugs, 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.

Warning

The `MAX_FILE_SIZE` is advisory to the browser. It is easy to circumvent this maximum. So don't count on it that the browser obeys you wish! The PHP-settings for maximum-size, however, cannot be fooled.

Variables defined for uploaded files differs depends on PHP version and configuration. Following variables will be defined within the destination script upon a successful upload. When `track_vars` is enabled, `$HTTP_POST_FILES/$_FILES` array is initialized. Finally, related variables may be initialized as globals when `register_globals` is turned on. However, use of globals is not recommended anymore.

Note: `track_vars` is always on from PHP 4.0.3. From PHP 4.1.0 or later, `$_FILES` may be used instead of `$HTTP_POST_FILES`. `$_FILES` is always global, so `global` should not be used for `$_FILES` in function scope.

`$HTTP_POST_FILES/$_FILES` is provided to contain the uploaded file information.

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 script 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.

Note: PHP 4.1.0 or later supports a short track variable `$_FILES`. PHP 3 does not support `$HTTP_POST_FILES`.

When `register_globals` is turned on in `php.ini` the available variables are as follows. Note that the following variable names assume the use of the file upload name 'userfile', as used in the example script 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".

Note: `register_globals = On` is not recommended for security and performance reason.

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 4 greater than 4.0.2. See the function entries for `is_uploaded_file()` and `move_uploaded_file()`.

```

<?php
// In PHP 4.1.0 or later, $_FILES should be used instead of $HTTP_POST_FILES.
if (is_uploaded_file($HTTP_POST_FILES['userfile']['tmp_name'])) {
    copy($HTTP_POST_FILES['userfile']['tmp_name'], "/place/to/put/uploaded/file");
} else {
    echo "Possible file upload attack. Filename: " . $HTTP_POST_FILES['userfile']['name'];
}
/* ...or... */
move_uploaded_file($HTTP_POST_FILES['userfile']['tmp_name'], "/place/to/put/uploaded/file");
?>

```

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 `$HTTP_POST_FILES['userfile']['size']` variable to throw away any files that are either too small or too big. You could use the `$HTTP_POST_FILES['userfile']['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` ini-setting. The default is 2 Megabytes.

If memory limit is enabled, larger `memory_limit` may be needed. Make sure to set `memory_limit` large enough.

If `max_execution_time` is set too small, script execution may be exceeded the value. Make sure to set `max_execution_time` large enough.

If `post_max_size` set too small, large files cannot be uploaded. Make sure to set `post_max_size` large enough.

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

Multiple files can be uploaded using different name for input.

It is also 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 `$HTTP_POST_FILES['userfile']`, `$HTTP_POST_FILES['userfile']['name']`, and `$HTTP_POST_FILES['userfile']['size']` will be initialized. (As well as in `$_FILES` for PHP 4.1.0 or later, `$HTTP_POST_VARS` in PHP 3. When `register_globals` is on, globals for uploaded files are also initialized). 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, `$HTTP_POST_FILES['userfile']['name'][0]` would contain the value `review.html`, and `$HTTP_POST_FILES['userfile']['name'][1]` would contain the value `xwp.out`. Similarly, `$HTTP_POST_FILES['userfile']['size'][0]` would contain `review.html`'s filesize, and so forth.

```
$HTTP_POST_FILES['userfile']['name'][0],
$HTTP_POST_FILES['userfile']['tmp_name'][0],
$HTTP_POST_FILES['userfile']['size'][0], and
$HTTP_POST_FILES['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.php
```

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

Inside your put.php 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
$file = fopen ("http://www.example.com/", "r");
if (!$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
$file = fopen ("ftp://ftp.example.com/incoming/outputfile", "w");
if (!$file) {
    echo "<p>Unable to open remote file for writing.\n";
```

```
        exit;
    }
    /* Write the data here. */
    fputs ($file, $HTTP_SERVER_VARS['HTTP_USER_AGENT'] . "\n");
    fclose ($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` `php.ini` directive as well as through the corresponding `"php_value 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` `php.ini` directive or the corresponding `"php_value 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).

Note: There are other extensions that provide persistent connections, such as the IMAP extension.

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

Why?

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

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

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

Warning

There are a couple of additional caveats to keep in mind when using persistent connections. One is that when using table locking on a persistent connection, if the script for whatever reason cannot release the lock, then subsequent scripts using the same connection will block indefinitely and may require that you either restart the httpd server or the database server. Another is that when using transactions, a transaction block will also carry over to the next script which uses that connection if script execution ends before the transaction block does. In either case, you can use `register_shutdown_function()` to register a simple cleanup function to unlock your tables or roll back your transactions. Better yet, avoid the problem entirely by not using persistent connections in scripts which use table locks or transactions (you can still use them elsewhere).

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!

See also `fbsql_pconnect()`, `ibase_pconnect()`, `ifx_pconnect()`, `imap_popen()`, `ingres_pconnect()`, `msql_pconnect()`, `mssql_pconnect()`, `mysql_pconnect()`, **OCIPLogon()**, `odbc_pconnect()`, **Ora_pLogon()**, `pfsockopen()`, `pg_pconnect()`, and `sybase_pconnect()`.

Chapter 23. Safe Mode

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

The configuration directives that control safe mode are:

```
safe_mode = Off
safe_mode_gid = 0
safe_mode_include_dir =
safe_mode_exec_dir =
open_basedir =
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars = LD_LIBRARY_PATH
disable_functions =
```

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

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

Running this `script.php`

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

results in this error when safe mode is enabled:

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

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

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

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

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

You can also disable individual functions. Note that the disable_functions directive can not be used outside of the php.ini file which means that you cannot disable functions on a per-virtualhost or per-directory basis in your httpd.conf file. If we add this to our php.ini file:

```
disable_functions readfile,system
```

Then we get this output:

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

Functions restricted/disabled by safe mode

This is a still probably incomplete and possibly incorrect listing of the functions limited by safe mode.

Table 23-1. Safe mode limited functions

Function	Limitations
dbmopen()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
dbase_open()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
filepro()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

Function	Limitations
<code>filepro_rowcount()</code>	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
<code>filepro_retrieve()</code>	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
<code>ifx_*</code>	<code>sql_safe_mode</code> restrictions, (!= safe mode)
<code>ingres_*</code>	<code>sql_safe_mode</code> restrictions, (!= safe mode)
<code>mysql_*</code>	<code>sql_safe_mode</code> restrictions, (!= safe mode)
<code>pg_loimport()</code>	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
<code>posix_mkfifo()</code>	Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
<code>putenv()</code>	Obeys the <code>safe_mode_protected_env_vars</code> and <code>safe_mode_allowed_env_vars</code> ini-directives. See also the documentation on <code>putenv()</code>
<code>move_uploaded_file()</code>	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
<code>chdir()</code>	Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
<code>dl()</code>	This function is disabled in safe mode.
backtick operator	This function is disabled in safe mode.
<code>shell_exec()</code> (functional equivalent of backticks)	This function is disabled in safe mode.
<code>exec()</code>	You can only execute executables within the <code>safe_mode_exec_dir</code> . For practical reasons it's currently not allowed to have <code>..</code> components in the path to the executable.
<code>system()</code>	You can only execute executables within the <code>safe_mode_exec_dir</code> . For practical reasons it's currently not allowed to have <code>..</code> components in the path to the executable.
<code>passthru()</code>	You can only execute executables within the <code>safe_mode_exec_dir</code> . For practical reasons it's currently not allowed to have <code>..</code> components in the path to the executable.
<code>popen()</code>	You can only execute executables within the <code>safe_mode_exec_dir</code> . For practical reasons it's currently not allowed to have <code>..</code> components in the path to the executable.

Function	Limitations
mkdir()	Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
rmdir()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
rename()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
unlink()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
copy()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed. (on <i>source</i> and <i>target</i>)
chgrp()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
chown()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.
chmod()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. In addition, you cannot set the SUID, SGID and sticky bits
touch()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed.
symlink()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed. (note: only the target is checked)

Function	Limitations
link()	Checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID as the script that is being executed. (note: only the target is checked)
getallheaders()	In safe mode, headers beginning with 'authorization' (case-insensitive) will not be returned. Warning: this is broken with the aol-server implementation of getallheaders()!
Any function that uses php4/main/fopen_wrappers.c	??

Part IV. Function Reference

I. Apache-specific Functions

apache_child_terminate (PHP 4 >= 4.0.5)

Terminate apache process after this request

bool **apache_child_terminate** (void) \linebreak

apache_child_terminate() will register the Apache process executing the current PHP request for termination once execution of PHP code it is completed. It may be used to terminate a process after a script with high memory consumption has been run as memory will usually only be freed internally but not given back to the operating system.

Note: The availability of this feature is controlled by the `php.ini` directive `apache.child_terminate`, which is set to `off` by default.

This feature is also not available on multithreaded versions of apache like the win32 version.

See also `exit()`.

apache_lookup_uri (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

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

object **apache_lookup_uri** (string filename) \linebreak

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 >= 4.0.0)

Get and set apache request notes

```
string apache_note ( string note_name [, string note_value]) \linebreak
```

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`.

apache_setenv (PHP 4 >= 4.2.0)

Set an Apache subprocess_env variable

```
int apache_setenv ( string variable, string value [, bool walk_to_top]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ascii2ebcdic (PHP 3>= 3.0.17)

Translate string from ASCII to EBCDIC

```
int ascii2ebcdic ( string ascii_str) \linebreak
```

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) \linebreak
```

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()`

getallheaders (PHP 3, PHP 4 >= 4.0.0)

Fetch all HTTP request headers

array **getallheaders** (void) \linebreak

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 >= 4.0.0)

Perform an Apache sub-request

int **virtual** (string filename) \linebreak

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.

II. Array Functions

Introduction

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.

Please see the Arrays section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Requirements

These functions are available as part of the standard module, which is always available.

Installation

There is no installation needed to use these functions, they are part of the PHP core.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

`CASE_UPPER` and `CASE_LOWER` are used with the `array_change_key_case()` function. They respectively

are used for changing a string to upper case or lower case.

See Also

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

array (unknown)

Create an array

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

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);
```

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);
```

will display :

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

See also `array_pad()`, `list()`, and `range()`.

array_change_key_case (PHP 4 >= 4.2.0)

Returns an array with all string keys lowercased or uppercased

array **array_change_key_case** (array input [, int case]) \linebreak

array_change_key_case() changes the keys in the *input* array to be all lowercase or uppercase. The change depends on the last optional *case* parameter. You can pass two constants there, `CASE_UPPER` and `CASE_LOWER`. The default is `CASE_LOWER`. The function will leave number indices as is.

Example 1. array_change_key_case() example

```
$input_array = array("FirSt" => 1, "SecOnd" => 4);
print_r(array_change_key_case($input_array, CASE_UPPER));
```

The printout of the above program will be:

```
Array
(
    [FIRST] => 1
    [SECOND] => 2
)
```

array_chunk (PHP 4 >= 4.2.0)

Split an array into chunks

array **array_chunk** (array input, int size [, bool preserve_keys]) \linebreak

array_chunk() splits the array into several arrays with *size* values in them. You may also have an array with less values at the end. You get the arrays as members of a multidimensional array indexed with numbers starting from zero.

By setting the optional *preserve_keys* parameter to `TRUE`, you can force PHP to preserve the original keys from the input array. If you specify `FALSE` new number indices will be used in each resulting array with indices starting from zero. The default is `FALSE`.

Example 1. array_chunk() example

```
$input_array = array('a', 'b', 'c', 'd', 'e');
print_r(array_chunk($input_array, 2));
print_r(array_chunk($input_array, 2, TRUE));
```

The printout of the above program will be:

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )
)
```

```

    )
    [1] => Array
    (
        [0] => c
        [1] => d
    )

    [2] => Array
    (
        [0] => e
    )
)
Array
(
    [0] => Array
    (
        [0] => a
        [1] => b
    )

    [1] => Array
    (
        [2] => c
        [3] => d
    )

    [2] => Array
    (
        [4] => e
    )
)

```

array_count_values (PHP 4 >= 4.0.0)

Counts all the values of an array

array **array_count_values** (array input) \linebreak

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

Example 1. array_count_values() example

```
$array = array (1, "hello", 1, "world", "hello");
print_r(array_count_values ($array));
```

The printout of the above program will be:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

array_diff (PHP 4)

Computes the difference of arrays

array **array_diff** (array array1, array array2 [, array ...]) \linebreak

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

This makes `$result` have array ("blue");. Multiple occurrences in `$array1` are all treated the same way.

Note: Two elements are considered equal if and only if `(string) $elem1 === (string) $elem2`. In words: when the string representation is the same.

Warning

This was broken in PHP 4.0.4!

See also `array_intersect()`.

array_fill (PHP 4 >= 4.2.0)

Fill an array with values

array **array_fill** (int *start_index*, int *num*, mixed *value*) \linebreak

array_fill() fills an array with *num* entries of the value of the *value* parameter, keys starting at the *start_index* parameter.

Example 1. array_fill() example

```
$a = array_fill(5, 6, 'banana');
```

\$a now has the following entries using `print_r()`:

```
Array
(
    [5] => banana
    [6] => banana
    [7] => banana
    [8] => banana
    [9] => banana
    [10] => banana
)
```

array_filter (PHP 4 >= 4.0.6)

Filters elements of an array using a callback function

array **array_filter** (array *input* [, mixed *callback*]) \linebreak

array_filter() returns an array containing all the elements of *input* filtered according a callback function. If the *input* is an associative array the keys are preserved.

Example 1. array_filter() example

```
function odd($var) {
    return ($var % 2 == 1);
}

function even($var) {
    return ($var % 2 == 0);
}

$array1 = array ("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);
$array2 = array (6, 7, 8, 9, 10, 11, 12);

echo "Odd :\n";
print_r(array_filter($array1, "odd"));
echo "Even:\n";
print_r(array_filter($array2, "even"));
```

The printout of the program above will be:

```
Odd :
Array
(
    [a] => 1
    [c] => 3
    [e] => 5
)
Even:
Array
(
    [0] => 6
    [2] => 8
    [4] => 10
    [6] => 12
)
```

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

Users may not change the array itself from the callback function. e.g. Add/delete an element, unset the array that **array_filter()** is applied to. If the array is changed, the behavior of this function is undefined.

See also `array_map()` and `array_reduce()`.

array_flip (PHP 4 >= 4.0.0)

Flip all the values of an array

array **array_flip** (array *trans*) \linebreak

array_flip() returns an array in flip order, i.e. keys from *trans* become values and *trans*'s values become keys.

Note that the values of *trans* need to be valid keys, i.e. they need to be either integer or string. A warning will be emitted if a value has the wrong type, and the key/value pair in question *will not be flipped*.

If a value has several occurrences, the latest key will be used as its values, and all others will be lost.

array_flip() returns `FALSE` if it fails.

Example 1. array_flip() example

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

Example 2. array_flip() example : collision

```
$trans = array ("a" => 1, "b" => 1, "c" => 2);
$trans = array_flip ($trans);
print_r($trans);
```

now \$trans is :

```
Array
(
    [1] => b
    [2] => c
)
```

array_intersect (PHP 4)

Computes the intersection of arrays

array **array_intersect** (array array1, array array2 [, array ...]) \linebreak

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  
    [0] => red  
)
```

Note: Two elements are considered equal if and only if `(string) $elem1 === (string) $elem2`. In words: when the string representation is the same.

Warning

This was broken in PHP 4.0.4!

See also `array_diff()`.

array_key_exists (PHP 4 >= 4.1.0)

Checks if the given key or index exists in the array

bool **array_key_exists** (mixed key, array search) \linebreak

array_key_exists() returns TRUE if the given *key* is set in the array. *key* can be any value possible for an array index.

Example 1. array_key_exists() example

```
$search_array = array("first" => 1, "second" => 4);
if (array_key_exists("first", $search_array)) {
    echo "The 'first' element is in the array";
}
```

Note: This name of this function is **key_exists()** in PHP version 4.0.6.

See also isset().

array_keys (PHP 4 >= 4.0.0)

Return all the keys of an array

array **array_keys** (array input [, mixed search_value]) \linebreak

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

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

```
$array = array ("blue", "red", "green", "blue", "blue");
print_r(array_keys ($array, "blue"));
```

```
$array = array ("color" => array("blue", "red", "green"), "size" => array("small", "medium", "large"));
print_r(array_keys ($array));
```

The printout of the program above will be:

```
Array
(
    [0] => 0
    [1] => color
)
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
Array
(
    [0] => color
    [1] => 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_map (PHP 4 >= 4.0.6)

Applies the callback to the elements of the given arrays

array **array_map** (mixed callback, array arr1 [, array arr2...]) \linebreak

array_map() returns an array containing all the elements of *arr1* after applying the callback function to each one. The number of parameters that the callback function accepts should match the number of arrays passed to the **array_map()**

Example 1. array_map() example

```
function cube($n) {
    return $n*$n*$n;
}

$a = array(1, 2, 3, 4, 5);
$b = array_map("cube", $a);
print_r($b);
```

This makes \$b have:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

Example 2. array_map() - using more arrays

```
function show_Spanish($n, $m) {
    return "The number $n is called $m in Spanish";
}

function map_Spanish($n, $m) {
    return array ($n => $m);
}

$a = array(1, 2, 3, 4, 5);
```

```

$b = array("uno", "dos", "tres", "cuatro", "cinco");

$c = array_map("show_Spanish", $a, $b);
print_r($c);

$d = array_map("map_Spanish", $a , $b);
print_r($d);

```

This results:

```

// printout of $c
Array
(
    [0] => The number 1 is called uno in Spanish
    [1] => The number 2 is called dos in Spanish
    [2] => The number 3 is called tres in Spanish
    [3] => The number 4 is called cuatro in Spanish
    [4] => The number 5 is called cinco in Spanish
)

// printout of $d
Array
(
    [0] => Array
        (
            [1] => uno
        )

    [1] => Array
        (
            [2] => dos
        )

    [2] => Array
        (
            [3] => tres
        )

    [3] => Array
        (
            [4] => cuatro
        )

    [4] => Array
        (
            [5] => cinco
        )
)

```

Usually when using two or more arrays, they should be of equal length because the callback function is applied in parallel to the corresponding elements. If the arrays are of unequal length, the shortest one will be extended with empty elements.

An interesting use of this function is to construct an array of arrays, which can be easily performed by using `NULL` as the name of the callback function

Example 3. Creating an array of arrays

```
$a = array(1, 2, 3, 4, 5);
$b = array("one", "two", "three", "four", "five");
$c = array("uno", "dos", "tres", "cuatro", "cinco");

$d = array_map(null, $a, $b, $c);
print_r($d);
```

The printout of the program above will be:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => one
            [2] => uno
        )

    [1] => Array
        (
            [0] => 2
            [1] => two
            [2] => dos
        )

    [2] => Array
        (
            [0] => 3
            [1] => three
            [2] => tres
        )

    [3] => Array
        (
            [0] => 4
```

```

        [1] => four
        [2] => cuatro
    )

[4] => Array
(
    [0] => 5
    [1] => five
    [2] => cinco
)
)

```

See also `array_filter()` and `array_reduce()`.

array_merge (PHP 4 >= 4.0.0)

Merge two or more arrays

array **array_merge** (array array1, array array2 [, array ...]) \linebreak

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);
$result = array_merge ($array1, $array2);

```

The `$result` will be:

```

Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => trapezoid
)

```

```

    [4] => 4
)

```

See also `array_merge_recursive()`.

array_merge_recursive (PHP 4)

Merge two or more arrays recursively

array **array_merge_recursive** (array array1, array array2 [, array ...]) \linebreak

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);

```

The `$result` will be:

```

Array
(
    [color] => Array
        (
            [favorite] => Array
                (
                    [0] => red
                    [1] => green
                )
            [0] => blue
        )
    [0] => 5
    [1] => 10
)

```

)

See also `array_merge()`.

array_multisort (PHP 4 >= 4.0.0)

Sort multiple or multi-dimensional arrays

bool **array_multisort** (array ar1 [, mixed arg [, mixed ... [, array ...]]) \linebreak

array_multisort() can be used to sort several arrays at once or a multi-dimensional array according by one of 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 sorting 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 >= 4.0.0)

Pad array to the specified length with a value

```
array array_pad ( array input, int pad_size, mixed pad_value) \linebreak
```

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 >= 4.0.0)

Pop the element off the end of array

mixed **array_pop** (array array) \linebreak

array_pop() pops and returns the last value of the *array*, shortening the *array* by one element. If *array* is empty (or is not an array), `NULL` will be returned.

Example 1. array_pop() example

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

After this, `$stack` will have only 3 elements:

```
Array  
(  
    [0] => orange  
    [1] => banana  
    [2] => apple  
)
```

and `raspberry` will be assigned to `$fruit`.

Warning

This function may return Boolean `FALSE`, but may also return a non-Boolean value which evaluates to `FALSE`, such as `0` or `""`. Please read the section on Booleans for more information. Use the `===` operator for testing the return value of this function.

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

array_push (PHP 4 >= 4.0.0)

Push one or more elements onto the end of array

int **array_push** (array *array*, mixed *var* [, mixed ...]) \linebreak

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

```
$stack = array ("orange", "banana");
array_push ($stack, "apple", "raspberry");
```

This example would result in *\$stack* having the following elements:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

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*]) \linebreak

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 `srand()` to seed the random number generator.

Example 1. array_rand() example

```
srand ((float) 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_reduce (PHP 4 >= 4.0.5)

Iteratively reduce the array to a single value using a callback function

mixed **array_reduce** (array input, mixed callback [, int initial]) \linebreak

array_reduce() applies iteratively the *callback* function to the elements of the array *input*, so as to reduce the array to a single value. If the optional *initial* is available, it will be used at the beginning of the process, or as a final result in case the array is empty.

Example 1. array_reduce() example

```
function rsum($v, $w) {
    $v += $w;
    return $v;
}

function rmul($v, $w) {
    $v *= $w;
    return $v;
}

$a = array(1, 2, 3, 4, 5);
$x = array();
$b = array_reduce($a, "rsum");
```

```
$c = array_reduce($a, "rmul", 10);
$d = array_reduce($x, "rsum", 1);
```

This will result in `$b` containing 15, `$c` containing 1200 (= 1*2*3*4*5*10), and `$d` containing 1.

See also `array_filter()` and `array_map()`.

array_reverse (PHP 4 >= 4.0.0)

Return an array with elements in reverse order

array **array_reverse** (array array [, bool preserve_keys]) \linebreak

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 both `$result` and `$result_keyed` have the same elements, but note the difference between the keys. The printout of `$result` and `$result_keyed` will be:

```
Array
(
    [0] => Array
        (
            [0] => green
            [1] => red
        )

    [1] => 4
    [2] => php
)
Array
(
    [2] => Array
        (
            [0] => green
            [1] => red
        )
)
```

```

    [1] => 4
    [0] => php
)

```

Note: The second parameter was added in PHP 4.0.3.

array_search (PHP 4 >= 4.0.5)

Searches the array for a given value and returns the corresponding key if successful

mixed **array_search** (mixed *needle*, array *haystack* [, bool *strict*]) \linebreak

Searches *haystack* for *needle* and returns the key if it is found in the array, FALSE otherwise.

Note: Prior to PHP 4.2.0, **array_search()** returns NULL on failure instead of FALSE.

If the optional third parameter *strict* is set to TRUE then the **array_search()** will also check the types of the *needle* in the *haystack*.

Warning

This function may return Boolean FALSE, but may also return a non-Boolean value which evaluates to FALSE, such as 0 or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

See also array_keys() and in_array().

array_shift (PHP 4 >= 4.0.0)

Shift an element off the beginning of array

mixed **array_shift** (array *array*) \linebreak

array_shift() shifts the first value of the *array* off and returns it, shortening the *array* by one element and moving everything down. If *array* is empty (or is not an array), NULL will be returned.

Example 1. array_shift() example

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

This would result in `$stack` having 3 elements left:

```
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
```

and `orange` will be assigned to `$fruit`.

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

array_slice (PHP 4 >= 4.0.0)

Extract a slice of the array

array **array_slice** (array *array*, int *offset* [, int *length*]) \linebreak

array_slice() returns the sequence of elements from the array *array* as 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*.

Note that **array_slice()** will ignore array keys, and will calculate offsets and lengths based on the actual positions of elements within 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 >= 4.0.0)

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

array **array_splice** (array input, int offset [, int length [, array replacement]]) \linebreak

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. It returns an array containing the extracted elements.

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")

$input = array ("red", "green", "blue", "yellow");
array_splice ($input, 1, -1);
// $input is now array ("red", "yellow")

$input = array ("red", "green", "blue", "yellow");
array_splice ($input, 1, count($input), "orange");
// $input is now array ("red", "orange")

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

```

See also `array_slice()`.

array_sum (PHP 4 >= 4.0.4)

Calculate the sum of values in an array.

mixed **array_sum** (array array) \linebreak

array_sum() returns the sum of values in an array as an integer or float.

Example 1. array_sum() examples

```

$a = array(2, 4, 6, 8);
echo "sum(a) = ".array_sum($a)."\n";

$b = array("a"=>1.2, "b"=>2.3, "c"=>3.4);
echo "sum(b) = ".array_sum($b)."\n";

```

The printout of the program above will be:

```

sum(a) = 20
sum(b) = 6.9

```

Note: PHP versions prior to 4.0.6 modified the passed array itself and converted strings to numbers (which most of the time converted them to zero, depending on their value).

array_unique (PHP 4)

Removes duplicate values from an array

array **array_unique** (array array) \linebreak

array_unique() takes input *array* and returns a new array without duplicate values.

Note that keys are preserved. **array_unique()** sorts the values treated as string at first, then will keep the first key encountered for every value, and ignore all following keys. It does not mean that the key of the first related value from the unsorted *array* will be kept.

Note: Two elements are considered equal if and only if (string) \$elem1 === (string) \$elem2. In words: when the string representation is the same.

The first element will be used.

Warning

This was broken in PHP 4.0.4!

Example 1. array_unique() example

```
$input = array ("a" => "green", "red", "b" => "green", "blue", "red");
$result = array_unique ($input);
print_r($result);
```

This will output:

```
Array
(
    [b] => green
    [1] => blue
    [2] => red
)
```

Example 2. array_unique() and types

```
$input = array (4, "4", "3", 4, 3, "3");
$result = array_unique ($input);
var_dump($result);
```

The printout of the program above will be (PHP 4.0.6):

```
array(2) {
  [3]=>
  int(4)
  [4]=>
  int(3)
}
```

array_unshift (PHP 4 >= 4.0.0)

Prepend one or more elements to the beginning of array

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

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

```
$queue = array ("orange", "banana");
array_unshift ($queue, "apple", "raspberry");
```

This would result in *\$queue* having the following elements:

```
Array
(
    [0] => apple
    [1] => raspberry
    [2] => orange
    [3] => banana
)
```

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

array_values (PHP 4 >= 4.0.0)

Return all the values of an array

array **array_values** (array input) \linebreak

array_values() returns all the values from the *input* array.

Example 1. array_values() example

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

This will output:

```
Array
(
    [0] => XL
    [1] => 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;
}
```

See also array_keys().

array_walk (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Apply a user function to every member of an array

```
int array_walk ( array array, string func [, mixed userdata] ) \linebreak
```

Applies the user-defined function named by *func* to each element of *array*. *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. *func* must be a user-defined function, and can't be a native PHP function. Thus, you can't use **array_walk()** straight with **str2lower()**, you must build a user-defined function with it first, and pass this function as argument.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

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.

Modifying the array from inside *func* may cause unpredictable behavior.

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.

Users may not change the array itself from the callback function. e.g. Add/delete element, unset the array that **array_walk()** is applied to. If the array is changed, the behavior of this function is undefined.

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";
}

echo "Before ...:\n";
array_walk ($fruits, 'test_print');
reset ($fruits);
array_walk ($fruits, 'test_alter', 'fruit');
echo "... and after:\n";
reset ($fruits);
array_walk ($fruits, 'test_print');
```

The printout of the program above will be:

```
Before ...:
d. lemon
a. orange
b. banana
c. apple
... and after:
d. fruit: lemon
a. fruit: orange
b. fruit: banana
c. fruit: apple
```

See also `each()` and `list()`.

arsort (PHP 3, PHP 4 >= 4.0.0)

Sort an array in reverse order and maintain index association

```
void arsort ( array array [, int sort_flags]) \linebreak
```

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:

```
a = orange
d = lemon
b = banana
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 >= 4.0.0)

Sort an array and maintain index association

```
void asort ( array array [, int sort_flags]) \linebreak
```

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:

```

c = apple
b = banana
d = lemon
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 >= 4.0.0)

Create array containing variables and their values

array **compact** (mixed varname [, mixed ...]) \linebreak

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 >= 4.0.0)

Count elements in a variable

int **count** (mixed *var*) \linebreak

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

If *var* is not an array, 1 will be returned (exception: `count (NULL)` equals 0).

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.

Please see the Arrays section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Example 1. count() example

```

$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count ($a);
// $result == 3

$b[0] = 7;
$b[5] = 9;
$b[10] = 11;
$result = count ($b);
// $result == 3;

```

Note: The `sizeof()` function is an alias for `count()`.

See also `is_array()`, `isset()`, and `strlen()`.

current (PHP 3, PHP 4 >= 4.0.0)

Return the current element in an array

mixed **current** (array array) \linebreak

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 >= 4.0.0)

Return the current key and value pair from an array and advance the array cursor

array **each** (array array) \linebreak

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. You have to use `reset()` if you want to traverse the array again using `each`.

See also `key()`, `list()`, `current()`, `reset()`, `next()`, `prev()`, and **`foreach()`**.

end (PHP 3, PHP 4 >= 4.0.0)

Set the internal pointer of an array to its last element

mixed **end** (array array) \linebreak

end() advances *array*'s internal pointer to the last element, and returns that element.

Example 1. A simple end() example

```
<?php
    $fruits = array('apple', 'banana', 'cranberry');
    print end($fruits); // cranberry
?>
```

See also `current()`, `each()`, `next()`, and `reset()`.

extract (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Import variables into the current symbol table from an array

int **extract** (array var_array [, int extract_type [, string prefix]]) \linebreak

This function is used to import variables from an array into the current symbol table. It takes an 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.

Note: EXTR_IF_EXISTS and EXTR_PREFIX_IF_EXISTS was introduced in version 4.2.0.

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 was added in PHP 4.0.5.

EXTR_IF_EXISTS

Only overwrite the variable if it already exists in the current symbol table, otherwise do nothing. This is useful for defining a list of valid variables and then extracting only those variables you have defined out of `$_REQUEST`, for example. This flag was added in PHP 4.2.0.

EXTR_PREFIX_IF_EXISTS

Only create prefixed variable names if the non-prefixed version of the same variable exists in the current symbol table. This flag was added in PHP 4.2.0.

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, EXTR_PREFIX_INVALID or EXTR_PREFIX_IF_EXISTS. 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 the 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 unless you use `EXTR_PREFIX_ALL` or `EXTR_PREFIX_INVALID`.

See also `compact()`.

in_array (PHP 4 >= 4.0.0)

Return `TRUE` if a value exists in an array

```
bool in_array ( mixed needle, array haystack [, bool strict] )\linebreak
```

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()` function will also check the types of the *needle* in the *haystack*.

Note: If *needle* is a string, the comparison is done in a case-sensitive manner.

Note: In PHP versions before 4.2.0 *needle* was not allowed to be an array.

Example 1. `in_array()` example

```
$os = array ("Mac", "NT", "Irix", "Linux");
if (in_array ("Irix", $os)) {
    print "Got Irix";
}
if (in_array ("mac", $os)) {
    print "Got mac";
}
```

The second condition fails because `in_array()` is case-sensitive, so the program above will display:

```
Got Irix
```

Example 2. `in_array()` with strict example

```
<?php
$a = array('1.10', 12.4, 1.13);

if (in_array('12.4', $a, TRUE))
    echo "'12.4' found with strict check\n";
if (in_array(1.13, $a, TRUE))
    echo "1.13 found with strict check\n";
?>
```

This will display:

```
1.13 found with strict check
```

Example 3. in_array() with an array as needle

```

<?php
$a = array(array('p', 'h'), array('p', 'r'), 'o');

if (in_array(array ('p', 'h'), $a))
    echo "'ph' is found\n";
if (in_array(array ('f', 'i'), $a))
    echo "'fi' is not found\n";
if (in_array('o', $a))
    echo "'o' is found\n";
?>

// This will output:

'ph' is found
'o' is found

```

See also `array_search()`.

key (PHP 3, PHP 4 >= 4.0.0)

Fetch a key from an associative array

mixed **key** (array array) \linebreak

key() returns the index element of the current array position.

See also `current()` and `next()`.

krsort (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sort an array by key in reverse order

int **krsort** (array array [, int sort_flags]) \linebreak

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:

```

d = lemon
c = apple
b = banana
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 >= 4.0.0)

Sort an array by key

```
int ksort ( array array [, int sort_flags]) \linebreak
```

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:

```
a = orange
b = banana
c = apple
d = lemon
```

You may modify the behavior of the sort using the optional parameter *sort_flags*, for details see `sort()`.

See also `asort()`, `arsort()`, `krsort()`, `uksort()`, `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 ( mixed ... ) \linebreak
```

Like `array()`, this is not really a function, but a language construct. **list()** is used to assign a list of variables in one operation.

Note: **list()** only works on numerical arrays and assumes the numerical indices start at 0.

Example 1. list() examples

```
<?php

$info = array('coffee', 'brown', 'caffeine');

// Listing all the variables
list($drink, $color, $power) = $info;
print "$drink is $color and $power makes it special.\n";

// Listing some of them
list($drink, , $power) = $info;
print "$drink has $power.\n";
```

```
// Or let's skip to only the third one
list( , , $power) = $info;
print "I need $power!\n";

?>
```

Example 2. An example use of list()

```
<table>
  <tr>
    <th>Employee name</th>
    <th>Salary</th>
  </tr>

<?php
$result = mysql_query ("SELECT id, name, salary FROM employees", $conn);
while (list ($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()`, `array()` and `extract()`.

natcasesort (PHP 4 >= 4.0.0)

Sort an array using a case insensitive "natural order" algorithm

```
void natcasesort ( array array) \linebreak
```

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://naturalordersort.org/>) page.

See also `sort()`, `natsort()`, `strnatcmp()`, and `strnatcasecmp()`.

natsort (PHP 4 >= 4.0.0)

Sort an array using a "natural order" algorithm

void **natsort** (array array) \linebreak

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://naturalordersort.org/>) page.

See also `natcasesort()`, `strnatcmp()`, and `strnatcasecmp()`.

next (PHP 3, PHP 4 >= 4.0.0)

Advance the internal array pointer of an array

mixed **next** (array array) \linebreak

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 >= 4.0.0)

Get the current element from an array

mixed **pos** (array array) \linebreak

This is an alias for `current()`.

See also `end()`, `next()`, `prev()`, and `reset()`.

prev (PHP 3, PHP 4 >= 4.0.0)

Rewind the internal array pointer

mixed **prev** (array array) \linebreak

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 >= 4.0.0)

Create an array containing a range of elements

array **range** (mixed low, mixed high) \linebreak

range() returns an array of elements from *low* to *high*, inclusive. If `low > high`, the sequence will be from high to low.

Example 1. range() examples

```
foreach(range(0, 9) as $number) {
    echo $number;
}
foreach(range('a', 'z') as $letter) {
    echo $letter;
}
foreach(range('z', 'a') as $letter) {
    echo $letter;
}
```

Note: Prior to version 4.1.0 the **range()** function only generated incrementing integer arrays. Support for character sequences and decrementing arrays was added in 4.1.0.

Example 2. Simulating decrementing ranges and character sequences

```
# array_reverse can be used to flip the order of a range
foreach(array_reverse(range(0,9)) as $number) {
    echo $number;
}

# array_map() can be used to turn integers into characters using chr()
foreach(array_map('chr', range(ord('a'),ord('z')))) as $character) {
    echo $character;
}
```

See `shuffle()` for another example of its use.

reset (PHP 3, PHP 4 >= 4.0.0)

Set the internal pointer of an array to its first element

mixed **reset** (array array) \linebreak

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 >= 4.0.0)

Sort an array in reverse order

void **rsort** (array array [, int sort_flags]) \linebreak

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:

```
0 = orange
1 = lemon
2 = banana
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 >= 4.0.0)

Shuffle an array

void **shuffle** (array array) \linebreak

This function shuffles (randomizes the order of the elements in) an array. You must use `srand()` to seed this function.

Example 1. shuffle() example

```
$numbers = range (1,20);
srand ((float)microtime()*1000000);
shuffle ($numbers);
while (list (, $number) = each ($numbers)) {
    echo "$number ";
}
```

See also `arsort()`, `asort()`, `ksort()`, `rsort()`, `sort()`, and `usort()`.

sizeof (PHP 3, PHP 4 >= 4.0.0)

Get the number of elements in variable

int **sizeof** (mixed var) \linebreak

The **sizeof()** function is an alias for `count()`.

See also `count()`.

sort (PHP 3, PHP 4 >= 4.0.0)

Sort an array

void **sort** (array array [, int sort_flags]) \linebreak

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."\n";
}
?>
```

This example would display:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

The fruits have been sorted in alphabetical order.

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 >= 4.0.0)

Sort an array with a user-defined comparison function and maintain index association

```
void uasort ( array array, function cmp_function) \linebreak
```

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

See also `usort()`, `uksort()`, `sort()`, `asort()`, `arsort()`, `ksort()`, and `rsort()`.

uksort (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Sort an array by keys using a user-defined comparison function

```
void uksort ( array array, function cmp_function) \linebreak
```

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

```

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

See also `usort()`, `uasort()`, `sort()`, `asort()`, `arsort()`, `ksort()`, `natsort()`, and `rsort()`.

usort (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Sort an array by values using a user-defined comparison function

```
void usort ( array array, string cmp_function) \linebreak
```

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
```

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

Example 3. usort() example using a member function of an object

```
class TestObj {
    var $name;

    function TestObj($name)
    {
        $this->name = $name;
    }

    /* This is the static comparing function: */
    function cmp_obj($a, $b)
    {
        $al = strtolower($a->name);
        $bl = strtolower($b->name);
        if ($al == $bl) return 0;
        return ($al > $bl) ? +1 : -1;
    }
}

$a[] = new TestObj("c");
$a[] = new TestObj("b");
$a[] = new TestObj("d");

uasort($a, array ("TestObj", "cmp_obj"));

foreach ($a as $item) {
    print $item->name."\n";
}
```

This example would display:

b
c
d

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 [deprecated]

Introduction

The `aspell()` functions allows you to check the spelling on a word and offer suggestions.

Requirements

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.

Installation

You need the aspell library, available from: <http://aspell.sourceforge.net/>.

See Also

See also pspell.

aspell_check (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Check a word [deprecated]

bool **aspell_check** (int dictionary_link, string word) \linebreak

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 >= 4.0.0)

Check a word without changing its case or trying to trim it [deprecated]

bool **aspell_check_raw** (int dictionary_link, string word) \linebreak

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_new (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Load a new dictionary [deprecated]

int **aspell_new** (string master [, string personal]) \linebreak

aspell_new() opens up a new dictionary and returns the dictionary link identifier for use in other aspell functions. Returns FALSE on error.

Example 1. aspell_new()

```
$aspell_link = aspell_new("english");
```

aspell_suggest (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Suggest spellings of a word [deprecated]

array **aspell_suggest** (int dictionary_link, string word) \linebreak

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");

    foreach ($suggestions as $suggestion) {
        echo "Possible spelling: $suggestion<br>\n";
    }
}
```

IV. BCMath Arbitrary Precision Mathematics Functions

Introduction

For arbitrary precision mathematics PHP offers the Binary Calculator which supports numbers of any size and precision, represented as strings.

Requirements

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.

Installation

In PHP 4, these functions are only available if PHP was configured with `--enable-bcmath`. In PHP 3, these functions are only available if PHP was not configured with `--disable-bcmath`.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

This extension does not define any constants.

bcadd (PHP 3, PHP 4 >= 4.0.0)

Add two arbitrary precision numbers

string **bcadd** (string left_operand, string right_operand [, int scale]) \linebreak

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 >= 4.0.0)

Compare two arbitrary precision numbers

int **bccomp** (string left_operand, string right_operand [, int scale]) \linebreak

Compares the *left_operand* to the *right_operand* and returns the result as an integer. The optional *scale* parameter is used to set the number of digits after the decimal place which will be used in the comparison. The return value is 0 if the two operands are equal. If the *left_operand* is larger than the *right_operand* the return value is +1 and if the *left_operand* is less than the *right_operand* the return value is -1.

bcdiv (PHP 3, PHP 4 >= 4.0.0)

Divide two arbitrary precision numbers

string **bcdiv** (string left_operand, string right_operand [, int scale]) \linebreak

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 >= 4.0.0)

Get modulus of an arbitrary precision number

string **bcmod** (string left_operand, string modulus) \linebreak

Get the modulus of the *left_operand* using *modulus*.

See also bcdiv().

bcmul (PHP 3, PHP 4 >= 4.0.0)

Multiply two arbitrary precision number

string **bcmul** (string left_operand, string right_operand [, int scale]) \linebreak

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 bcdiv().

bcpow (PHP 3, PHP 4 >= 4.0.0)

Raise an arbitrary precision number to another

string **bcpow** (string x, string y [, int scale]) \linebreak

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 >= 4.0.0)

Set default scale parameter for all bc math functions

string **bcscale** (int scale) \linebreak

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 >= 4.0.0)

Get the square root of an arbitrary precision number

string **bcsqrt** (string operand [, int scale]) \linebreak

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 >= 4.0.0)

Subtract one arbitrary precision number from another

string **bcsub** (string *left_operand*, string *right_operand* [, int *scale*]) \linebreak

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. Bzip2 Compression Functions

Introduction

The bzip2 functions are used to transparently read and write bzip2 (.bz2) compressed files.

Requirements

This module uses the functions of the bzip2 (<http://sources.redhat.com/bzip2/>) library by Julian Seward

Installation

Bzip2 support in PHP is not enabled by default. You will need to use the `--with-bz2` configuration option when compiling PHP to enable bzip2 support. This module requires bzip2/libbzip2 version `>= 1.0.x`.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension defines one resource type: a file pointer identifying the bz2-file to work on.

Predefined Constants

This extension does not define any constants.

Examples

This example opens a temporary file and writes a test string to it, then prints out the contents of the file.

Example 1. Small bzip2 Example

```
<?php
$filename = "/tmp/testfile.bz2";
```

```
$str = "This is a test string.\n";

// open file for writing
$bz = bzopen($filename, "w");

// write string to file
bzwrite($bz, $str);

// close file
bzclose($bz);

// open file for reading
$bz = bzopen($filename, "r");

// read 10 characters
print bzread($bz, 10);

// output until end of the file (or the next 1024 char) and close it.
print bzread($bz);

bzclose($bz);

?>
```

bzclose (PHP 4 >= 4.0.4)

Close a bzip2 file pointer

```
int bzclose ( resource bz) \linebreak
```

Closes the bzip2 file referenced by the pointer *bz*.

Returns `TRUE` on success, `FALSE` on failure.

The file pointer must be valid, and must point to a file successfully opened by `bzopen()`.

See also `bzopen()`.

bzcompress (PHP 4 >= 4.0.4)

Compress a string into bzip2 encoded data

```
string bzcompress ( string source [, int blocksize [, int workfactor]]) \linebreak
```

`bzcompress()` compresses the *source* string and returns it as bzip2 encoded data.

The optional parameter *blocksize* specifies the blocksize used during compression and should be a number from 1 to 9 with 9 giving the best compression, but using more resources to do so. *blocksize* defaults to 4.

The optional parameter *workfactor* controls how the compression phase behaves when presented with worst case, highly repetitive, input data. The value can be between 0 and 250 with 0 being a special case and 30 being the default value. Regardless of the *workfactor*, the generated output is the same.

Example 1. bzcompress() Example

```
<?php
$str = "sample data";
$bzstr = bzcompress($str, 9);
print( $bzstr );
?>
```

See also `bzdecompress()`.

bzdecompress (PHP 4 >= 4.0.4)

Decompresses bzip2 encoded data

string **bzdecompress** (string source [, int small]) \linebreak

bzdecompress() decompresses the *source* string containing bzip2 encoded data and returns it. If the optional parameter *small* is `TRUE`, an alternative decompression algorithm will be used which uses less memory (the maximum memory requirement drops to around 2300K) but works at roughly half the speed. See the bzip2 documentation (<http://sources.redhat.com/bzip2/>) for more information about this feature.

Example 1. bzdecompress()

```
<?php
$start_str = "This is not an honest face?";
$bzstr = bzcompress($start_str);

print( "Compressed String: " );
print( $bzstr );
print( "\n<br>\n" );

$str = bzdecompress($bzstr);
print( "Decompressed String: " );
print( $str );
print( "\n<br>\n" );
?>
```

See also `bzcompress()`.

bzerrno (PHP 4 >= 4.0.4)

Returns a bzip2 error number

int **bzerrno** (resource bz) \linebreak

Returns the error number of any bzip2 error returned by the file pointer *bz*.

See also `bzerror()` and `bzerrstr()`.

bzerror (PHP 4 >= 4.0.4)

Returns the bzip2 error number and error string in an array

array **bzerror** (resource bz) \linebreak

Returns the error number and error string, in an associative array, of any bzip2 error returned by the file pointer *bz*.

Example 1. bzerror() Example

```
<?php
$error = bzerror($bz);

echo $error["errno"];
echo $error["errstr"];
?>
```

See also bzerrno() and bzerrstr().

bzerrstr (PHP 4 >= 4.0.4)

Returns a bzip2 error string

string **bzerrstr** (resource *bz*) \linebreak

Returns the error string of any bzip2 error returned by the file pointer *bz*.

See also bzerrno() and bzerror().

bzflush (PHP 4 >= 4.0.4)

Force a write of all buffered data

int **bzflush** (resource *bz*) \linebreak

Forces a write of all buffered bzip2 data for the file pointer *bz*.

Returns TRUE on success, FALSE on failure.

See also bzread() and bzwrite().

bzopen (PHP 4 >= 4.0.4)

Open a bzip2 compressed file

resource **bzopen** (string *filename*, string *mode*) \linebreak

Opens a bzip2 (.bz2) file for reading or writing. *filename* is the name of the file to open. *mode* is similar to the fopen() function ('r' for read, 'w' for write, etc.).

If the open fails, the function returns FALSE, otherwise it returns a pointer to the newly opened file.

Example 1. bzopen() Example

```
<?php
$bz = bzopen("/tmp/foo.bz2", "r");
$decompressed_file = bzread($bz, filesize("/tmp/foo.bz2"));
bzclose($bz);

print( "The contents of /tmp/foo.bz2 are: " );
print( "\n<br>\n" );
print( $decompressed_file );
?>
```

See also bzclose().

bzread (PHP 4 >= 4.0.4)

Binary safe bzip2 file read

string **bzread** (resource bz [, int length]) \linebreak

bzread() reads up to *length* bytes from the bzip2 file pointer referenced by *bz*. Reading stops when *length* (uncompressed) bytes have been read or EOF is reached, whichever comes first. If the optional parameter *length* is not specified, **bzread()** will read 1024 (uncompressed) bytes at a time.

Example 1. bzread() Example

```
<?php
$bz = bzopen("/tmp/foo.bz2", "r");
$str = bzread($bz, 2048);
print( $str );
?>
```

See also bzwrite() and bzopen().

bzwrite (PHP 4 >= 4.0.4)

Binary safe bzip2 file write

int **bzwrite** (resource *bz*, string *data* [, int *length*]) \linebreak

bzwrite() writes the contents of the string *data* to the bzip2 file stream pointed to by *bz*. If the optional *length* argument is given, writing will stop after *length* (uncompressed) bytes have been written or the end of string is reached, whichever comes first.

Example 1. bzwrite() Example

```
<?php
$str = "uncompressed data";
$bz = bzopen("/tmp/foo.bz2", "w");
bzwrite($bz, $str, strlen($str));
bzclose($bz);
?>
```

See also `bzread()` and `bzopen()`.

VI. Calendar functions

Introduction

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 from January 1st, 4713 B.C. 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 Julian Day Count, visit http://serendipity.magnet.ch/hermetic/cal_stud/jdn.htm. 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.

Installation

To get these functions to work, you have to compile PHP with `--enable-calendar`.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either

been compiled into PHP or dynamically loaded at runtime.

CAL_GREGORIAN (integer)

CAL_JULIAN (integer)

CAL_JEWISH (integer)

CAL_FRENCH (integer)

CAL_NUM_CALS (integer)

CAL_DOW_DAYNO (integer)

CAL_DOW_SHORT (integer)

CAL_DOW_LONG (integer)

CAL_MONTH_GREGORIAN_SHORT (integer)

CAL_MONTH_GREGORIAN_LONG (integer)

CAL_MONTH_JULIAN_SHORT (integer)

CAL_MONTH_JULIAN_LONG (integer)

CAL_MONTH_JEWISH (integer)

CAL_MONTH_FRENCH (integer)

cal_days_in_month (PHP 4 >= 4.1.0)

Return the number of days in a month for a given year and calendar

```
int cal_days_in_month ( int calendar, int month, int year) \linebreak
```

This function will return the number of days in the *month* of *year* for the specified *calendar*.

See also `jdtounix()`.

cal_from_jd (PHP 4 >= 4.1.0)

Converts from Julian Day Count to a supported calendar and return extended information

```
array cal_from_jd ( int jd, int calendar) \linebreak
```

cal_info (PHP 4 >= 4.1.0)

Returns information about a particular calendar

```
array cal_info ( int calendar) \linebreak
```

cal_to_jd (PHP 4 >= 4.1.0)

Converts from a supported calendar to Julian Day Count

```
int cal_to_jd ( int calendar, int month, int day, int year) \linebreak
```

easter_date (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Get UNIX timestamp for midnight on Easter of a given year

```
int easter_date ( int year) \linebreak
```

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 >= 4.0.0)

Get number of days after March 21 on which Easter falls for a given year

```
int easter_days ( int year) \linebreak
```

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()`.

FrenchToJD (PHP 3, PHP 4 >= 4.0.0)

Converts a date from the French Republican Calendar to a Julian Day Count

`int frenchtojd (int month, int day, int year) \linebreak`

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.

GregorianToJD (PHP 3, PHP 4 >= 4.0.0)

Converts a Gregorian date to Julian Day Count

`int gregoriantojd (int month, int day, int year) \linebreak`

Valid Range for Gregorian Calendar 4714 B.C. to 9999 A.D.

Although this function 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";
?>
```

JDDayOfWeek (PHP 3, PHP 4 >= 4.0.0)

Returns the day of the week

mixed **jddayofweek** (int julianday, int mode) \linebreak

Returns the day of the week. Can return a string or an integer 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)

JDMonthName (PHP 3, PHP 4 >= 4.0.0)

Returns a month name

string **jdmonthname** (int julianday, int mode) \linebreak

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	Values
0	Gregorian - abbreviated	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
1	Gregorian	January, February, March, April, May, June, July, August, September, October, November, December
2	Julian - abbreviated	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
3	Julian	January, February, March, April, May, June, July, August, September, October, November, December

Mode	Meaning	Values
4	Jewish	Tishri, Heshvan, Kislev, Tevet, Shevat, AdarI, AdarII, Nisan, Iyyar, Sivan, Tammuz, Av, Elul
5	French Republican	Vendemiaire, Brumaire, Frimaire, Nivose, Pluviouse, Ventose, Germinal, Floreal, Prairial, Messidor, Thermidor, Fructidor, Extra

JDToFrench (PHP 3, PHP 4 >= 4.0.0)

Converts a Julian Day Count to the French Republican Calendar

string **jdtofrrench** (int juliandaycount) \linebreak

Converts a Julian Day Count to the French Republican Calendar.

JDToGregorian (PHP 3, PHP 4 >= 4.0.0)

Converts Julian Day Count to Gregorian date

string **jdtogregorian** (int julianday) \linebreak

Converts Julian Day Count to a string containing the Gregorian date in the format of "month/day/year".

JDToJewish (PHP 3, PHP 4 >= 4.0.0)

Converts a Julian Day Count to the Jewish Calendar

string **jdtojewish** (int julianday) \linebreak

Converts a Julian Day Count the the Jewish Calendar.

JDToJulian (PHP 3, PHP 4 >= 4.0.0)

Converts a Julian Day Count to a Julian Calendar Date

string **jdtojulian** (int julianday) \linebreak

Converts Julian Day Count to a string containing the Julian Calendar Date in the format of "month/day/year".

jdtounix (PHP 4 >= 4.0.0)

Convert Julian Day to UNIX timestamp

`int jdtounix (int jday) \linebreak`

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 $2440588 \leq jday \leq 2465342$)

See also `unixtojd()`.

JewishToJD (PHP 3, PHP 4 >= 4.0.0)

Converts a date in the Jewish Calendar to Julian Day Count

`int jewishtojd (int month, int day, int year) \linebreak`

Although this function 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.

JulianToJD (PHP 3, PHP 4 >= 4.0.0)

Converts a Julian Calendar date to Julian Day Count

`int juliantojd (int month, int day, int year) \linebreak`

Valid Range for Julian Calendar 4713 B.C. to 9999 A.D.

Although this function 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.

Caution

Remember, the current calendar system being used worldwide is the Gregorian calendar. `gregoriantojd()` can be used to convert such dates to their Julian Day count.

unixtojd (PHP 4 >= 4.0.0)

Convert UNIX timestamp to Julian Day

int **unixtojd** ([int *timestamp*]) \linebreak

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()`.

VII. CCVS API Functions

Introduction

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.

Note: CCVS has been discontinued by Red Hat and there are no plans to issue further keys or support contracts. Those looking for a replacement can consider MCVE by Main Street Softworks (<http://www.mcve.com/>) as a potential replacement. It is similar in design and has documented PHP support!

Installation

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.)

See Also

Additional information about CCVS can be found at <http://www.redhat.com/products/ccvs>. RedHat maintains slightly outdated but still useful documentation at <http://www.redhat.com/products/ccvs/support/CCVS3.3docs/ProgPHP.html>.

ccvs_add (PHP 4 >= 4.0.2)

Add data to a transaction

string **ccvs_add** (string session, string invoice, string argtype, string argval) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_auth (PHP 4 >= 4.0.2)

Perform credit authorization test on a transaction

string **ccvs_auth** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_command (PHP 4 >= 4.0.2)

Performs a command which is peculiar to a single protocol, and thus is not available in the general CCVS API

string **ccvs_command** (string session, string type, string argval) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_count (PHP 4 >= 4.0.2)

Find out how many transactions of a given type are stored in the system

int **ccvs_count** (string session, string type) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_delete (PHP 4 >= 4.0.2)

Delete a transaction

string **ccvs_delete** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_done (PHP 4 >= 4.0.2)

Terminate CCVS engine and do cleanup work

string **ccvs_done** (string sess) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_init (PHP 4 >= 4.0.2)

Initialize CCVS for use

string **ccvs_init** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_lookup (PHP 4 >= 4.0.2)

Look up an item of a particular type in the database #

string **ccvs_lookup** (string session, string invoice, int inum) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_new (PHP 4 >= 4.0.2)

Create a new, blank transaction

string **ccvs_new** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_report (PHP 4 >= 4.0.2)

Return the status of the background communication process

string **ccvs_report** (string session, string type) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_return (PHP 4 >= 4.0.2)

Transfer funds from the merchant to the credit card holder

string **ccvs_return** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_reverse (PHP 4 >= 4.0.2)

Perform a full reversal on an already-processed authorization

string **ccvs_reverse** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_sale (PHP 4 >= 4.0.2)

Transfer funds from the credit card holder to the merchant

string **ccvs_sale** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_status (PHP 4 >= 4.0.2)

Check the status of an invoice

string **ccvs_status** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_textvalue (PHP 4 >= 4.0.2)

Get text return value for previous function call

string **ccvs_textvalue** (string session) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ccvs_void (PHP 4 >= 4.0.2)

Perform a full reversal on a completed transaction

string **ccvs_void** (string session, string invoice) \linebreak

Warning

This function is currently not documented; only the argument list is available.

VIII. COM support functions for Windows

Introduction

COM is a technology which allows the reuse of code written in any language (by any language) using a standard calling convention and hiding behind APIs the implementation details such as what machine the Component is stored on and the executable which houses it. It can be thought of as a super Remote Procedure Call (RPC) mechanism with some basic object roots. It separates implementation from interface.

COM encourages versioning, separation of implementation from interface and hiding the implementation details such as executable location and the language it was written in.

COM functions are only available on the Windows version of PHP.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either

been compiled into PHP or dynamically loaded at runtime.

CLSCTX_INPROC_SERVER (integer)

CLSCTX_INPROC_HANDLER (integer)

CLSCTX_LOCAL_SERVER (integer)

CLSCTX_REMOTE_SERVER (integer)

CLSCTX_SERVER (integer)

CLSCTX_ALL (integer)

VT_NULL (integer)

VT_EMPTY (integer)

VT_UI1 (integer)

VT_I2 (integer)

VT_I4 (integer)

VT_R4 (integer)

VT_R8 (integer)

VT_BOOL (integer)

VT_ERROR (integer)

VT_CY (integer)

VT_DATE (integer)

(<http://www.microsoft.com/Com/resources/comdocs.asp>) or perhaps take a look at Don Box's Yet Another COM Library (YACL) (<http://www.developmentor.com/dbox/yacl.htm>)

COM (unknown)

COM class

Synopsis

```
$obj = new COM("server.object")
```

The COM class provides a framework to integrate (D)COM components into your php scripts.

```
string COM::COM ( string module_name [, string server_name [, int codepage]]) \linebreak
```

COM class constructor. Parameters:

`module_name`

name or class-id of the requested component.

`server_name`

name of the DCOM server from which the component should be fetched. If NULL, localhost is assumed. To allow DCOM `com.allow_dcom` has to be set to TRUE in `php.ini`.

`codepage`

specifies the codepage that is used to convert php-strings to unicode-strings and vice versa. Possible values are CP_ACP, CP_MACCP, CP_OEMCP, CP_SYMBOL, CP_THREAD_ACP, CP_UTF7 and CP_UTF8.

Example 1. COM example (1)

```
// starting word
$word = new COM("word.application") or die("Unable to instanciate Word");
print "Loaded Word, version {$word->Version}\n";

//bring it to front
$word->Visible = 1;

//open an empty document
$word->Documents->Add();

//do some weird stuff
$word->Selection->TypeText("This is a test...");
$word->Documents[1]->SaveAs("Useless test.doc");

//closing word
$word->Quit();
```

```
//free the object
$word->Release();
$word = null;
```

Example 2. COM example (2)

```
$conn = new COM("ADODB.Connection") or die("Cannot start ADO");
$conn->Open("Provider=SQLOLEDB; Data Source=localhost;
Initial Catalog=database; User ID=user; Password=password");

$rs = $conn->Execute("SELECT * FROM sometable"); // Recordset

$num_columns = $rs->Fields->Count();
echo $num_columns . "\n";

for ($i=0; $i < $num_columns; $i++)
{
    $fld[$i] = $rs->Fields($i);
}

$rowcount = 0;
while (!$rs->EOF)
{
    for ($i=0; $i < $num_columns; $i++)
    {
        echo $fld[$i]->value . "\t";
    }
    echo "\n";
    $rowcount++; // increments rowcount
    $rs->MoveNext();
}

$rs->Close();
$conn->Close();

$rs->Release();
$conn->Release();

$rs = null;
$conn = null;
```


VARIANT (unknown)

VARIANT class

Synopsis

```
$vVar = new VARIANT($var)
```

A simple container to wrap variables into VARIANT structures.

```
string VARIANT::VARIANT ( [mixed value [, int type [, int codepage]]) \linebreak
```

VARIANT class constructor. Parameters:

value

initial value. if omitted an VT_EMPTY object is created.

type

specifies the content type of the VARIANT object. Possible values are VT_UI1, VT_UI2, VT_UI4, VT_I1, VT_I2, VT_I4, VT_R4, VT_R8, VT_INT, VT_UINT, VT_BOOL, VT_ERROR, VT_CY, VT_DATE, VT_BSTR, VT_DECIMAL, VT_UNKNOWN, VT_DISPATCH and VT_VARIANT. These values are mutual exclusive, but they can be combined with VT_BYREF to specify being a value. If omitted, the type of *value* is used. Consult the msdn library for additional information.

codepage

specifies the codepage that is used to convert php-strings to unicode-strings and vice versa. Possible values are CP_ACP, CP_MACCP, CP_OEMCP, CP_SYMBOL, CP_THREAD_ACP, CP_UTF7 and CP_UTF8.

com_addrf (PHP 4 >= 4.1.0)

Increases the components reference counter.

```
void com_addrf ( void) \linebreak
```

Increases the components reference counter.

com_get (PHP 3 >= 3.0.3, PHP 4 >= 4.0.5)

Gets the value of a COM Component's property

mixed **com_get** (resource *com_object*, string *property*) \linebreak

Returns the value of the *property* of the COM component referenced by *com_object*. Returns `FALSE` on error.

com_invoke (PHP 3>= 3.0.3)

Calls a COM component's method.

mixed **com_invoke** (resource *com_object*, string *function_name* [, mixed function parameters, ...]) \linebreak

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_isenum (PHP 4 >= 4.1.0)

Grabs an IEnumVariant

void **com_isenum** (object *com_module*) \linebreak

Warning

This function is currently not documented; only the argument list is available.

com_load (PHP 3>= 3.0.3)

Creates a new reference to a COM component

string **com_load** (string *module name* [, string *server name* [, int *codepage*]]) \linebreak

com_load() creates a new COM component and returns a reference to it. Returns `FALSE` on failure. Possible values for *codepage* are `CP_ACP`, `CP_MACCP`, `CP_OEMCP`, `CP_SYMBOL`, `CP_THREAD_ACP`, `CP_UTF7` and `CP_UTF8`.

com_load_typelib (PHP 4 >= 4.1.0)

Loads a Typelib

void **com_load_typelib** (string *typelib_name* [, int *case_insensitive*]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

com_propget (PHP 3>= 3.0.3, PHP 4 >= 4.0.5)

Gets the value of a COM Component's property

mixed **com_propget** (resource com_object, string property) \linebreak

This function is an alias for com_get().

com_propput (PHP 3>= 3.0.3, PHP 4 >= 4.0.5)

Assigns a value to a COM component's property

void **com_propput** (resource com_object, string property, mixed value) \linebreak

This function is an alias for com_set().

com_propset (PHP 3>= 3.0.3, PHP 4 >= 4.0.5)

Assigns a value to a COM component's property

void **com_propset** (resource com_object, string property, mixed value) \linebreak

This function is an alias for com_set().

com_release (PHP 4 >= 4.1.0)

Decreases the components reference counter.

void **com_release** (void) \linebreak

Decreases the components reference counter.

com_set (PHP 3 >= 3.0.3, PHP 4 >= 4.0.5)

Assigns a value to a COM component's property

void **com_set** (resource com_object, string property, mixed value) \linebreak

Sets the value of the *property* of the COM component referenced by *com_object*. Returns the newly set value if succeeded, `FALSE` on error.

IX. Class/Object Functions

Introduction

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).

Examples

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 >= 4.0.0)

Call a user method on an specific object [deprecated]

mixed **call_user_method** (string *method_name*, object *obj* [, mixed *parameter* [, mixed ...]]) \linebreak

Warning

The **call_user_method()** function is deprecated as of PHP 4.1.0, use the **call_user_func()** variety with the `array(&$obj, "method_name")` syntax instead.

Calls 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";
    }
}

$centry = new Country("Peru","pe");

echo "* Calling the object method directly\n";
$centry->print_info();

echo "\n* Calling the same method indirectly\n";
call_user_method ("print_info", $centry, "\t");
?>
```

See also `call_user_func_array()`, `call_user_func()`, and `call_user_method_array()`.

call_user_method_array (PHP 4 >= 4.0.5)

Call a user method given with an array of parameters [deprecated]

mixed **call_user_method_array** (string *method_name*, object *obj* [, array *paramarr*]) \linebreak

Warning

The **call_user_method_array()** function is deprecated as of PHP 4.1.0, use the **call_user_func_array()** variety with the `array(&$obj, "method_name")` syntax instead.

Calls the method referred by *method_name* from the user defined *obj* object, using the parameters in *paramarr*.

See also: `call_user_func_array()`, `call_user_func()`, `call_user_method()`.

Note: This function was added to the CVS code after release of PHP 4.0.4pl1

class_exists (PHP 4 >= 4.0.0)

Checks if the class has been defined

bool **class_exists** (string *class_name*) \linebreak

This function returns `TRUE` if the class given by *class_name* has been defined, `FALSE` otherwise.

get_class (PHP 4 >= 4.0.0)

Returns the name of the class of an object

string **get_class** (object *obj*) \linebreak

This function returns the name of the class of which the object *obj* is an instance. Returns `FALSE` if *obj* is not an object.

Note: **get_class()** returns a user defined class name in lowercase. A class defined in a PHP extension is returned in its original notation.

See also `get_parent_class()`, `gettype()`, and `is_subclass_of()`.

get_class_methods (PHP 4 >= 4.0.0)

Returns an array of class methods' names

array **get_class_methods** (mixed *class_name*) \linebreak

This function returns an array of method names defined for the class specified by *class_name*.

Note: As of PHP 4.0.6, you can specify the object itself instead of *class_name*. For example:

```
$class_methods = get_class_methods($my_class); // see below the full example
```

Example 1. get_class_methods() example

```
<?php

class myclass {
    // constructor
    function myclass() {
        return(true);
    }

    // method 1
    function myfunc1() {
        return(true);
    }

    // method 2
    function myfunc2() {
        return(true);
    }
}

$my_object = new myclass();

$class_methods = get_class_methods(get_class($my_object));

foreach ($class_methods as $method_name) {
    echo "$method_name\n";
}

?>
```

Will produce:

```
myclass  
myfunc1  
myfunc2
```

See also `get_class_vars()` and `get_object_vars()`.

get_class_vars (PHP 4 >= 4.0.0)

Returns an array of default properties of the class

array **get_class_vars** (string class_name) \linebreak

This function will return an associative array of default properties of the class. The resulting array elements are in the form of *varname => value*.

Note: Uninitialized class variables will not be reported by **get_class_vars()**.

Example 1. get_class_vars() example

```
<?php  
  
class myclass {  
  
    var $var1; // this has no default value...  
    var $var2 = "xyz";  
    var $var3 = 100;  
  
    // constructor  
    function myclass() {  
        return(true);  
    }  
  
}  
  
$my_class = new myclass();
```

```

$class_vars = get_class_vars(get_class($my_class));

foreach ($class_vars as $name => $value) {
    echo "$name : $value\n";
}

?>

```

Will produce:

```

var2 : xyz
var3 : 100

```

See also `get_class_methods()`, `get_object_vars()`

get_declared_classes (PHP 4 >= 4.0.0)

Returns an array with the name of the defined classes

array **get_declared_classes** (void) \linebreak

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`).

Also note that depending on what libraries you have compiled into PHP, additional classes could be present. This means that you will not be able to define your own classes using these names. There is a list of predefined classes in the Predefined Classes section of the appendices.

get_object_vars (PHP 4 >= 4.0.0)

Returns an associative array of object properties

array **get_object_vars** (object obj) \linebreak

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);
    }
}

// "$label" is declared but not defined
$p1 = new Point2D(1.233, 3.445);
print_r(get_object_vars($p1));

$p1->setLabel("point #1");
print_r(get_object_vars($p1));

?>
```

The printout of the above program will be:

```
Array
(
    [x] => 1.233
    [y] => 3.445
)

Array
(
    [x] => 1.233
    [y] => 3.445
    [label] => point #1
)
```

)

See also `get_class_methods()` and `get_class_vars()`!

get_parent_class (PHP 4 >= 4.0.0)

Retrieves the parent class name for object or class

string **get_parent_class** (mixed *obj*) \linebreak

If *obj* is an object, returns the name of the parent class of the class of which *obj* is an instance.

If *obj* is a string, returns the name of the parent class of the class with that name. This functionality was added in PHP 4.0.5.

See also `get_class()` and `is_subclass_of()`

is_a (PHP 4 >= 4.2.0)

Returns true if the object is of this class or has this class as one of its parents

bool **is_a** (object *object*, string *class_name*) \linebreak

This function returns `TRUE` if the object is of this class or has this class as one of its parents, `FALSE` otherwise.

See also `get_class()`, `get_parent_class()`, and `is_subclass_of()`.

is_subclass_of (PHP 4 >= 4.0.0)

Returns true if the object has this class as one of its parents

bool **is_subclass_of** (object *object*, string *class_name*) \linebreak

This function returns `TRUE` if the object *object*, belongs to a class which is a subclass of *class_name*, `FALSE` otherwise.

See also `get_class()`, `get_parent_class()` and `is_a()`.

method_exists (PHP 4 >= 4.0.0)

Checks if the class method exists

bool **method_exists** (object object, string method_name) \linebreak

This function returns `TRUE` if the method given by *method_name* has been defined for the given *object*, `FALSE` otherwise.

X. ClibPDF functions

Introduction

ClibPDF lets you create PDF documents with PHP. It is available for download from FastIO (<http://www.fastio.com/>), but requires that you purchase a license for commercial use. ClibPDF functionality and API are similar to PDFlib.

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.

A nice feature of ClibPDF (and PDFlib) is the ability 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 feature can also be simulated by `pdf_translate()` when using the PDFlib functions.)

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.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either

been compiled into PHP or dynamically loaded at runtime.

CPDF_PM_NONE (integer)

CPDF_PM_OUTLINES (integer)

CPDF_PM_THUMBS (integer)

CPDF_PM_FULLSCREEN (integer)

CPDF_PL_SINGLE (integer)

CPDF_PL_1COLUMN (integer)

CPDF_PL_2LCOLUMN (integer)

CPDF_PL_2RCOLUMN (integer)

Examples

Example 1. Simple ClibPDF Example

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842, 1.0);
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);
    }
}
```

```

$time = getdate();

/* draw hour hand */
cpdf_save($pdf);
cpdf_rotate($pdf, -(($time['minutes']/60.0) + $time['hours'] - 3.0) * 30.0);
cpdf_moveto($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, -(($time['seconds']/60.0) + $time['minutes'] - 15.0) * 6.0);
cpdf_moveto($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, -(($time['seconds'] - 15.0) * 6.0));
cpdf_moveto($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);
?>

```

See Also

See also the PDFlib extension documentation.

cpdf_add_annotation (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Adds annotation

```
void cpdf_add_annotation ( int pdf document, float llx, float lly, float urx, float ury, string title, string content [, int mode]) \linebreak
```

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.

cpdf_add_outline (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Adds bookmark for current page

```
void cpdf_add_outline ( int pdf document, string text) \linebreak
```

The **cpdf_add_outline()** function adds a bookmark with text *text* that points to the current page.

Example 1. Adding a page outline

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
// ...
// some drawing
// ...
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

cpdf_arc (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draws an arc

```
void cpdf_arc ( int pdf document, float x-coor, float y-coor, float radius, float start, float end [, int mode])
\linebreak
```

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_begin_text (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Starts text section

```
void cpdf_begin_text ( int pdf document) \linebreak
```

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_circle (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draw a circle

```
void cpdf_circle ( int pdf document, float x-coor, float y-coor, float radius [, int mode]) \linebreak
```

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_clip (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Clips to current path

```
void cpdf_clip ( int pdf document) \linebreak
```

The **cpdf_clip()** function clips all drawing to the current path.

cpdf_close (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Closes the pdf document

```
void cpdf_close ( int pdf document) \linebreak
```

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_closepath (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Close path

```
void cpdf_closepath ( int pdf document) \linebreak
```

The **cpdf_closepath()** function closes the current path.

cpdf_closepath_fill_stroke (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Close, fill and stroke current path

```
void cpdf_closepath_fill_stroke ( int pdf document) \linebreak
```

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_closepath_stroke (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Close path and draw line along path

void **cpdf_closepath_stroke** (int pdf document) \linebreak

The **cpdf_closepath_stroke()** function is a combination of **cpdf_closepath()** and **cpdf_stroke()**. Then clears the path.

See also **cpdf_closepath()**, **cpdf_stroke()**.

cpdf_continue_text (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Output text in next line

void **cpdf_continue_text** (int pdf document, string text) \linebreak

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_curveto (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draws a curve

void **cpdf_curveto** (int pdf document, float x1, float y1, float x2, float y2, float x3, float y3 [, int mode]) \linebreak

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_lineto()**.

cpdf_end_text (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Ends text section

void **cpdf_end_text** (int pdf document) \linebreak

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_fill (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Fill current path

```
void cpdf_fill ( int pdf document) \linebreak
```

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 >= 4.0.0)

Fill and stroke current path

```
void cpdf_fill_stroke ( int pdf document) \linebreak
```

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_finalize (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Ends document

```
void cpdf_finalize ( int pdf document) \linebreak
```

The `cpdf_finalize()` function ends the document. You still have to call `cpdf_close()`

See also `cpdf_close()`.

cpdf_finalize_page (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Ends page

```
void cpdf_finalize_page ( int pdf document, int page number) \linebreak
```

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_global_set_document_limits (PHP 4 >= 4.0.0)

Sets document limits for any pdf document

```
void cpdf_global_set_document_limits ( int maxpages, int maxfonts, int maximages, int maxannotations, int maxobjects) \linebreak
```

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_import_jpeg (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Opens a JPEG image

```
int cpdf_import_jpeg ( int pdf document, string file name, float x-coor, float y-coor, float angle, float width, float height, float x-scale, float y-scale [, int mode]) \linebreak
```

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_lineto (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draws a line

void **cpdf_lineto** (int pdf document, float x-coor, float y-coor [, int mode]) \linebreak

The **cpdf_lineto()** 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_moveto (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets current point

void **cpdf_moveto** (int pdf document, float x-coor, float y-coor [, int mode]) \linebreak

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_newpath (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Starts a new path

void **cpdf_newpath** (int pdf document) \linebreak

The **cpdf_newpath()** starts a new path on the document given by the *pdf document* parameter.

cpdf_open (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Opens a new pdf document

int **cpdf_open** (int compression [, string filename]) \linebreak

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_output_buffer (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Outputs the pdf document in memory buffer

```
void cpdf_output_buffer ( int pdf document) \linebreak
```

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_page_init (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Starts new page

```
void cpdf_page_init ( int pdf document, int page number, int orientation, float height, float width [, float unit]) \linebreak
```

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_place_inline_image (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Places an image on the page

```
void cpdf_place_inline_image ( int pdf document, int image, float x-coor, float y-coor, float angle, float width, float height [, int mode]) \linebreak
```

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_rect (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draw a rectangle

void **cpdf_rect** (int pdf document, float x-coor, float y-coor, float width, float height [, int mode]) \linebreak

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_restore (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Restores formerly saved environment

void **cpdf_restore** (int pdf document) \linebreak

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_rlineto (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Draws a line

```
void cpdf_rlineto ( int pdf document, float x-coor, float y-coor [, int mode]) \linebreak
```

The **cpdf_rlineto()** 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_rmoveto (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Sets current point

```
void cpdf_rmoveto ( int pdf document, float x-coor, float y-coor [, int mode]) \linebreak
```

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_rotate (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets rotation

```
void cpdf_rotate ( int pdf document, float angle) \linebreak
```

The **cpdf_rotate()** function set the rotation in degrees to *angle*.

cpdf_rotate_text (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Sets text rotation angle

```
void cpdf_rotate_text ( int pdfdoc, float angle) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cpdf_save (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Saves current environment

```
void cpdf_save ( int pdf document) \linebreak
```

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_save_to_file (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Writes the pdf document into a file

```
void cpdf_save_to_file ( int pdf document, string filename) \linebreak
```

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_scale (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets scaling

```
void cpdf_scale ( int pdf document, float x-scale, float y-scale) \linebreak
```

The **cpdf_scale()** function set the scaling factor in both directions.

cpdf_set_action_url (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Sets hyperlink

```
void cpdf_set_action_url ( int pdfdoc, float xll, float yll, float xur, float yur, string url [, int mode]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cpdf_set_char_spacing (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets character spacing

```
void cpdf_set_char_spacing ( int pdf document, float space) \linebreak
```

The **cpdf_set_char_spacing()** function sets the spacing between characters.

See also **cpdf_set_word_spacing()**, **cpdf_set_leading()**.

cpdf_set_creator (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the creator field in the pdf document

```
void cpdf_set_creator ( string creator) \linebreak
```

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_current_page (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Sets current page

```
void cpdf_set_current_page ( int pdf document, int page number) \linebreak
```

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_set_font (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Select the current font face and size

```
void cpdf_set_font ( int pdf document, string font name, float size, string encoding) \linebreak
```


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_font_directories (PHP 4 >= 4.0.6)

Sets directories to search when using external fonts

```
void cpdf_set_font_directories ( int pdfdoc, string pfmdir, string pfbdir) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cpdf_set_font_map_file (PHP 4 >= 4.0.6)

Sets fontname to filename translation map when using external fonts

```
void cpdf_set_font_map_file ( int pdfdoc, string filename) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cpdf_set_horiz_scaling (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets horizontal scaling of text

```
void cpdf_set_horiz_scaling ( int pdf document, float scale) \linebreak
```

The **cpdf_set_horiz_scaling()** function sets the horizontal scaling to *scale* percent.

cpdf_set_keywords (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the keywords field of the pdf document

```
void cpdf_set_keywords ( string keywords) \linebreak
```

The **cpdf_set_keywords()** function sets the keywords of a pdf document.

See also `cpdf_set_title()`, `cpdf_set_creator()`, `cpdf_set_subject()`.

cpdf_set_leading (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets distance between text lines

```
void cpdf_set_leading ( int pdf document, float distance) \linebreak
```

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_page_animation (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Sets duration between pages

```
void cpdf_set_page_animation ( int pdf document, int transition, float duration) \linebreak
```

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_set_subject (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the subject field of the pdf document

```
void cpdf_set_subject ( string subject) \linebreak
```

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_text_matrix (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the text matrix

```
void cpdf_set_text_matrix ( int pdf document, array matrix) \linebreak
```

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 >= 4.0.0)

Sets text position

```
void cpdf_set_text_pos ( int pdf document, float x-coor, float y-coor [, int mode]) \linebreak
```

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_text_rendering (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Determines how text is rendered

```
void cpdf_set_text_rendering ( int pdf document, int mode) \linebreak
```

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_text_rise (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the text rise

void **cpdf_set_text_rise** (int pdf document, float value) \linebreak

The **cpdf_set_text_rise()** function sets the text rising to *value* units.

cpdf_set_title (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets the title field of the pdf document

void **cpdf_set_title** (string title) \linebreak

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_viewer_preferences (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

How to show the document in the viewer

void **cpdf_set_viewer_preferences** (int pdfdoc, array preferences) \linebreak

Warning

This function is currently not documented; only the argument list is available.

cpdf_set_word_spacing (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets spacing between words

void **cpdf_set_word_spacing** (int pdf document, float space) \linebreak

The **cpdf_set_word_spacing()** function sets the spacing between words.

See also **cpdf_set_char_spacing()**, **cpdf_set_leading()**.

cpdf_setdash (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets dash pattern

void **cpdf_setdash** (int pdf document, float white, float black) \linebreak

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_setflat (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets flatness

```
void cpdf_setflat ( int pdf document, float value) \linebreak
```

The **cpdf_setflat()** function set the flatness to a value between 0 and 100.

cpdf_setgray (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets drawing and filling color to gray value

```
void cpdf_setgray ( int pdf document, float gray value) \linebreak
```

The **cpdf_setgray()** function sets the current drawing and filling color to the given gray value.

See also `cpdf_setrgbcolor_stroke()`, `cpdf_setrgbcolor_fill()`.

cpdf_setgray_fill (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets filling color to gray value

```
void cpdf_setgray_fill ( int pdf document, float value) \linebreak
```

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 >= 4.0.0)

Sets drawing color to gray value

```
void cpdf_setgray_stroke ( int pdf document, float gray value) \linebreak
```

The **cpdf_setgray_stroke()** function sets the current drawing color to the given gray value.

See also `cpdf_setrgbcolor_stroke()`.

cpdf_setlinecap (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets linecap parameter

```
void cpdf_setlinecap ( int pdf document, int value) \linebreak
```

The **cpdf_setlinecap()** function set the linecap parameter between a value of 0 and 2. 0 = butt end, 1 = round, 2 = projecting square.

cpdf_setlinejoin (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets linejoin parameter

```
void cpdf_setlinejoin ( int pdf document, long value) \linebreak
```

The **cpdf_setlinejoin()** function set the linejoin parameter between a value of 0 and 2. 0 = miter, 1 = round, 2 = bevel.

cpdf_setlinewidth (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets line width

```
void cpdf_setlinewidth ( int pdf document, float width) \linebreak
```

The **cpdf_setlinewidth()** function set the line width to *width*.

cpdf_setmiterlimit (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets miter limit

```
void cpdf_setmiterlimit ( int pdf document, float value) \linebreak
```

The **cpdf_setmiterlimit()** function set the miter limit to a value greater or equal than 1.

cpdf_setrgbcolor (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets drawing and filling color to rgb color value

```
void cpdf_setrgbcolor ( int pdf document, float red value, float green value, float blue value) \linebreak
```

The **cpdf_setrgbcolor()** function sets the current drawing and filling color to the given rgb color value.

See also `cpdf_setrgbcolor_stroke()`, `cpdf_setrgbcolor_fill()`.

cpdf_setrgbcolor_fill (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets filling color to rgb color value

```
void cpdf_setrgbcolor_fill ( int pdf document, float red value, float green value, float blue value) \linebreak
```

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 >= 4.0.0)

Sets drawing color to rgb color value

```
void cpdf_setrgbcolor_stroke ( int pdf document, float red value, float green value, float blue value) \linebreak
```

The **cpdf_setrgbcolor_stroke()** function sets the current drawing color to the given rgb color value.

See also `cpdf_setrgbcolor_fill()`, `cpdf_setrgbcolor()`.

cpdf_show (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Output text at current position

```
void cpdf_show ( int pdf document, string text) \linebreak
```

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 >= 4.0.0)

Output text at position

```
void cpdf_show_xy ( int pdf document, string text, float x-coor, float y-coor [, int mode]) \linebreak
```

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_stringwidth (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Returns width of text in current font

float **cpdf_stringwidth** (int pdf document, string text) \linebreak

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_stroke (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Draw line along path

void **cpdf_stroke** (int pdf document) \linebreak

The **cpdf_stroke()** function draws a line along current path.

See also `cpdf_closepath()`, `cpdf_closepath_stroke()`.

cpdf_text (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Output text with parameters

void **cpdf_text** (int pdf document, string text, float x-coor, float y-coor [, int mode [, float orientation [, int alignmode]]]) \linebreak

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_translate (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Sets origin of coordinate system

void **cpdf_translate** (int pdf document, float x-coor, float y-coor [, int mode]) \linebreak

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.

XI. Crack functions

Introduction

These functions allow you to use the CrackLib library to test the 'strength' of a password. The 'strength' of a password is tested by that checks length, use of upper and lower case and checked against the specified CrackLib dictionary. CrackLib will also give helpful diagnostic messages that will help 'strengthen' the password.

Requirements

More information regarding CrackLib along with the library can be found at <http://www.users.dircon.co.uk/~crypto/>.

Installation

In order to use these functions, you must compile PHP with Crack support by using the `--with-crack[=DIR]` option.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

This extension does not define any constants.

Examples

This example shows how to open a CrackLib dictionary, test a given password, retrieve any diagnostic

messages, and close the dictionary.

Example 1. CrackLib example

```
<?php
// Open CrackLib Dictionary
$dictionary = crack_opendict('/usr/local/lib/pw_dict')
    or die('Unable to open CrackLib dictionary');

// Perform password check
$check = crack_check($dictionary, 'gx9A2s0x');

// Retrieve messages
$diag = crack_getlastmessage();
echo $diag; // 'strong password'

// Close dictionary
crack_closedict($dictionary);
?>
```

Note: If `crack_check()` returns `TRUE`, `crack_getlastmessage()` will return 'strong password'.

crack_check (PHP 4 >= 4.0.5)

Performs an obscure check with the given password

bool **crack_check** ([resource dictionary, string password]) \linebreak

Returns TRUE if *password* is strong, or FALSE otherwise.

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

crack_check() performs an obscure check with the given *password* on the specified *dictionary* . If *dictionary* is not specified, the last opened dictionary is used.

crack_closedict (PHP 4 >= 4.0.5)

Closes an open CrackLib dictionary

bool **crack_closedict** ([resource dictionary]) \linebreak

Returns TRUE on success, FALSE on failure.

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

crack_closedict() closes the specified *dictionary* identifier. If *dictionary* is not specified, the current dictionary is closed.

crack_getlastmessage (PHP 4 >= 4.0.5)

Returns the message from the last obscure check

string **crack_getlastmessage** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

crack_getlastmessage() returns the message from the last obscure check.

crack_opendict (PHP 4 >= 4.0.5)

Opens a new CrackLib dictionary

resource **crack_opendict** (string dictionary) \linebreak

Returns a dictionary resource identifier on success, or `FALSE` on failure.

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

crack_opendict() opens the specified CrackLib *dictionary* for use with `crack_check()`.

Note: Only one dictionary may be open at a time.

See also: `crack_check()`, and `crack_closedict()`.

XII. CURL, Client URL Library Functions

Introduction

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.

These functions have been added in PHP 4.0.2.

Requirements

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.

Installation

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.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either

been compiled into PHP or dynamically loaded at runtime.

CURLOPT_PORT (integer)

CURLOPT_FILE (integer)

CURLOPT_INFILE (integer)

CURLOPT_INFILESIZE (integer)

CURLOPT_URL (integer)

CURLOPT_PROXY (integer)

CURLOPT_VERBOSE (integer)

CURLOPT_HEADER (integer)

CURLOPT_HTTPHEADER (integer)

CURLOPT_NOPROGRESS (integer)

CURLOPT_NOBODY (integer)

CURLOPT_FAILONERROR (integer)

CURLOPT_UPLOAD (integer)

CURLOPT_POST (integer)

CURLOPT_FTPLISTONLY (integer)

CURLOPT_FTPAPPEND (integer)

CURLOPT_NETRC (integer)

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 `example.com` homepage into a file:

Example 1. Using PHP's CURL module to fetch the `example.com` homepage

```
<?php

$ch = curl_init ("http://www.example.com/");
$fp = fopen ("example_homepage.txt", "w");

curl_setopt ($ch, CURLOPT_FILE, $fp);
curl_setopt ($ch, CURLOPT_HEADER, 0);

curl_exec ($ch);
curl_close ($ch);
fclose ($fp);
?>
```


curl_close (PHP 4 >= 4.0.2)

Close a CURL session

```
void curl_close ( int ch) \linebreak
```

This function closes a CURL session and frees all resources. The CURL handle, *ch*, is also deleted.

curl_errno (PHP 4 >= 4.0.3)

Return an integer containing the last error number

```
int curl_errno ( int ch) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

curl_error (PHP 4 >= 4.0.3)

Return a string containing the last error for the current session

```
string curl_error ( int ch) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

curl_exec (PHP 4 >= 4.0.2)

Perform a CURL session

```
bool curl_exec ( int ch) \linebreak
```

This function 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*).

Tip: As with anything that outputs its result directly to the browser, you can use the output-control functions to capture the output of this function, and save it - for example - in a string.

curl_getinfo (PHP 4 >= 4.0.4)

Get information regarding a specific transfer

```
string curl_getinfo ( int ch, int opt) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

curl_init (PHP 4 >= 4.0.2)

Initialize a CURL session

```
int curl_init ( [string url]) \linebreak
```

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 ($ch, CURLOPT_URL, "http://www.example.com/");
curl_setopt ($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) \linebreak
```

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 behavior 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 to 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_SSL_VERIFYHOST*: Pass a long if CURL should verify the Common name of the peer certificate in the SSL handshake. A value of 1 denotes that we should check for the existence of the common name, a value of 2 denotes that we should make sure it matches the provided hostname.
- *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.
- *CURLOPT_RETURNTRANSFER*: Pass a non-zero value if you want CURL to directly return the transfer instead of printing it out directly.

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 value 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 cookie 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 other, more obscure, HTTP requests. Valid values are things like GET, POST, and so on; i.e. do not enter a whole HTTP request line here. For instance, entering 'GET /index.html HTTP/1.0\r\n\r\n' would be incorrect.

Note: Don't do this without making sure your server supports the command first.

- *CURLOPT_PROXY*: Give the name of the HTTP proxy to tunnel requests through.
- *CURLOPT_INTERFACE*: Pass the name of the outgoing network interface to use. This can be an interface name, an IP address or a host name.
- *CURLOPT_KRB4LEVEL*: Pass the KRB4 (Kerberos 4) security level. Anyone of the following strings (in order from least powerful, to most powerful): 'clear', 'safe', 'confidential', 'private'. If the string does not match one of these, then 'private' is used. If you set this to NULL, this disables KRB4 security. KRB4 security only works with FTP transactions currently.
- *CURLOPT_HTTPHEADER*: Pass an array of HTTP header fields to set.
- *CURLOPT_QUOTE*: Pass an array of FTP commands to perform on the server prior to the FTP request.
- *CURLOPT_POSTQUOTE*: Pass an array of FTP commands to execute on the server, after the FTP request has been performed.

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_version (PHP 4 >= 4.0.2)

Return the current CURL version

string **curl_version** (void) \linebreak

The **curl_version()** function returns a string containing the current CURL version.

XIII. Cybercash payment functions

Installation

These functions are only available if the interpreter has been compiled with the
`--with-cybercash=[DIR]`.

cybercash_base64_decode (PHP 4 >= 4.0.0)

base64 decode data for Cybercash

```
string cybercash_base64_decode ( string inbuff) \linebreak
```

cybercash_base64_encode (PHP 4 >= 4.0.0)

base64 encode data for Cybercash

```
string cybercash_base64_encode ( string inbuff) \linebreak
```

cybercash_decr (PHP 4 >= 4.0.0)

Cybercash decrypt

```
array cybercash_decr ( string wmk, string sk, string inbuff) \linebreak
```

The function returns an associative array with the elements "errcode" and, if "errcode" is FALSE, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_encr (PHP 4 >= 4.0.0)

Cybercash encrypt

```
array cybercash_encr ( string wmk, string sk, string inbuff) \linebreak
```

The function returns an associative array with the elements "errcode" and, if "errcode" is FALSE, "outbuff" (string), "outLth" (long) and "macbuff" (string).

XIV. Crédit Mutuel CyberMUT functions

Introduction

This extension allows you to process credit cards transactions using Crédit Mutuel CyberMUT system (http://www.creditmutuel.fr/centre_commercial/vendez_sur_internet.html).

CyberMUT is a popular Web Payment Service in France, provided by the Crédit Mutuel bank. If you are foreign in France, these functions will not be useful for you.

The use of these functions is almost identical to the original SDK functions, except for the parameters of return for `cybermut_creerformulairecm()` and `cybermut_crerreponsecm()`, which are returned directly by functions PHP, whereas they had passed in reference in the original functions.

These functions have been added in PHP 4.0.6.

Note: These functions only provide a link to CyberMUT SDK. Be sure to read the CyberMUT Developers Guide for full details of the required parameters.

Installation

These functions are only available if PHP has been compiled with the `--with-cybermut[=DIR]` option, where `DIR` is the location of `libcm-mac.a` and `cm-mac.h`. You will require the appropriate SDK for your platform, which may be sent to you after your CyberMUT's subscription (contact them via Web, or go to the nearest Crédit Mutuel).

cybermut_creerformulairecm (PHP 4 >= 4.0.5)

Generate HTML form of request for payment

```
string cybermut_creerformulairecm ( string url_CM, string version, string TPE, string montant, string ref_commande,
string texte_libre, string url_retour, string url_retour_ok, string url_retour_err, string langue, string code_societe,
string texte_bouton) \linebreak
```

cybermut_creerformulairecm() is used to generate the HTML form of request for payment.

Example 1. First step of payment (equiv cgi1.c)

```
<?php
// Directory where the keys are located
putenv("CMKEYDIR=/var/creditmut/cles");

// Version number
$VERSION="1.2";

$retour = cybermut_creerformulairecm(
    "https://www.creditmutuel.fr/test/telepaiement/paiement.cgi",
    $VERSION,
    "1234567890",
    "300FRF",
    $REFERENCE,
    $TEXTE_LIBRE,
    $URL_RETOUR,
    $URL_RETOUR_OK,
    $URL_RETOUR_ERR,
    "français",
    "company",
    "Paielement par carte bancaire");

echo $retour;
?>
```

See also `cybermut_testmac()` and `cybermut_creeerreponsecm()`.

cybermut_creeerreponsecm (PHP 4 >= 4.0.5)

Generate the acknowledgement of delivery of the confirmation of payment

```
string cybermut_creeerreponsecm ( string phrase) \linebreak
```

cybermut_creeerreponsecm() returns a string containing delivery acknowledgement message.

The parameter is "OK" if the message of confirmation of the payment was correctly identified by `cybermut_testmac()`. Any other chain is regarded as an error message.

See also `cybermut_creeerformulairecm()` and `cybermut_testmac()`.

cybermut_testmac (PHP 4 >= 4.0.5)

Make sure that there no was data diddling contained in the received message of confirmation

bool **cybermut_testmac** (string code_MAC, string version, string TPE, string cdate, string montant, string ref_commande, string texte_libre, string code-retour)\linebreak

cybermut_testmac() is used to make sure that there was not data diddling contained in the received message of confirmation. Pay attention to parameters *code-retour* and *texte-libre*, which cannot be evaluated as is, because of the dash. You must retrieve them by using:

```
<?php
    $code_retour=$HTTP_GET_VARS["code-retour"];
    $texte_libre=$HTTP_GET_VARS["texte-libre"];
?>
```

Example 1. Last step of payment (equiv cgi2.c)

```
<?php
// Make sure that Enable Track Vars is ON.
// Directory where are located the keys
putenv("CMKEYDIR=/var/creditmut/cles");

// Version number
$VERSION="1.2";

$texte_libre = $HTTP_GET_VARS["texte-libre"];
$code_retour = $HTTP_GET_VARS["code-retour"];

$mac_ok = cybermut_testmac($MAC,$VERSION,$TPE,$date,$montant,$reference,$texte_libre,$code_r

if ($mac_ok) {

    //
    // insert data processing here
    //
    //

    $result=cybermut_creeerreponsecm("OK");
```

```
} else {  
  $result=cybermut_creerreponsecm("Document Falsifie");  
}  
  
?>
```

See also `cybermut_creerformulairecm()` and `cybermut_creerreponsecm()`.

XV. Cyrus IMAP administration functions

Introduction

Warning

This function is currently not documented; only the argument list is available.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

CYRUS_CONN_NONSYNCLITERAL (integer)

CYRUS_CONN_INITIALRESPONSE (integer)

CYRUS_CALLBACK_NUMBERED (integer)

CYRUS_CALLBACK_NOLITERAL (integer)

cyrus_authenticate (PHP 4 >= 4.1.0)

Authenticate against a Cyrus IMAP server

```
bool cyrus_authenticate ( resource connection [, string mechlist [, string service [, string user [, int minssf [, int maxssf]]]]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cyrus_bind (PHP 4 >= 4.1.0)

Bind callbacks to a Cyrus IMAP connection

```
bool cyrus_bind ( resource connection, array callbacks) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cyrus_close (PHP 4 >= 4.1.0)

Close connection to a cyrus server

```
bool cyrus_close ( resource connection) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

cyrus_connect (PHP 4 >= 4.1.0)

Connect to a Cyrus IMAP server

resource **cyrus_connect** ([string host [, string port [, int flags]]]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

cyrus_query (PHP 4 >= 4.1.0)

Send a query to a Cyrus IMAP server

bool **cyrus_query** (resource connection, string query) \linebreak

Warning

This function is currently not documented; only the argument list is available.

cyrus_unbind (PHP 4 >= 4.1.0)

Unbind ...

bool **cyrus_unbind** (resource connection, string trigger_name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

XVI. Character type functions

Introduction

The functions provided by this extension check whether a character or string falls into a certain character class according to the current locale (see also `setlocale()`).

When called with an integer argument these functions behave exactly like their C counterparts from "ctype.h".

When called with a string argument they will check every character in the string and will only return `TRUE` if every character in the string matches the requested criteria.

Passing anything else but a string or integer will return `FALSE` immediately.

Requirements

None besides functions from the standard C library which are always available.

Installation

Beginning with PHP 4.2.0 these functions are enabled by default. For older versions you have to configure and compile PHP with `--enable-ctype`.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

This extension does not define any constants.

ctype_alnum (PHP 4 >= 4.0.4)

Check for alphanumeric character(s)

bool **ctype_alnum** (string text) \linebreak

Returns `TRUE` if every character in *text* is either a letter or a digit, `FALSE` otherwise. In the standard C locale letters are just `[A-Za-z]`. The function is equivalent to `(ctype_alpha($text) || ctype_digit($text))`.

See also `ctype_alpha()`, `ctype_digit()`, and `setlocale()`.

ctype_alpha (PHP 4 >= 4.0.4)

Check for alphabetic character(s)

bool **ctype_alpha** (string text) \linebreak

Returns `TRUE` if every character in *text* is a letter from the current locale, `FALSE` otherwise. In the standard C locale letters are just `[A-Za-z]` and **ctype_alpha()** is equivalent to `(ctype_upper($text) || ctype_lower($text))`, but other languages have letters that are considered neither upper nor lower case.

See also `ctype_upper()`, `ctype_lower()`, and `setlocale()`.

ctype_cntrl (PHP 4 >= 4.0.4)

Check for control character(s)

bool **ctype_cntrl** (string text) \linebreak

Returns `TRUE` if every character in *text* has a special control function, `FALSE` otherwise. Control characters are e.g. line feed, tab, esc.

ctype_digit (PHP 4 >= 4.0.4)

Check for numeric character(s)

bool **ctype_digit** (string text) \linebreak

Returns `TRUE` if every character in *text* is a decimal digit, `FALSE` otherwise.

See also `ctype_alnum()` and `ctype_xdigit()`.

ctype_graph (PHP 4 >= 4.0.4)

Check for any printable character(s) except space

bool **ctype_graph** (string text) \linebreak

Returns `TRUE` if every character in *text* is printable and actually creates visible output (no white space), `FALSE` otherwise.

See also `ctype_alnum()`, `ctype_print()`, and `ctype_punct()`.

ctype_lower (PHP 4 >= 4.0.4)

Check for lowercase character(s)

bool **ctype_lower** (string text) \linebreak

Returns `TRUE` if every character in *text* is a lowercase letter in the current locale.

See also `ctype_alpha()` and `ctype_upper()`.

ctype_print (PHP 4 >= 4.0.4)

Check for printable character(s)

bool **ctype_print** (string text) \linebreak

Returns `TRUE` if every character in *text* will actually create output (including blanks). Returns `FALSE` if *text* contains control characters or characters that do not have any output or control function at all.

See also `ctype_cntrl()`, `ctype_graph()`, and `ctype_punct()`.

ctype_punct (PHP 4 >= 4.0.4)

Check for any printable character which is not whitespace or an alphanumeric character

bool **ctype_punct** (string text) \linebreak

Returns `TRUE` if every character in *text* is printable, but neither letter, digit or blank, `FALSE` otherwise.

See also `ctype_cntrl()`, `ctype_graph()`, and **`ctype_punct()`**.

ctype_space (PHP 4 >= 4.0.4)

Check for whitespace character(s)

bool **ctype_space** (string text) \linebreak

Returns `TRUE` if every character in *text* creates some sort of white space, `FALSE` otherwise. Besides the blank character this also includes tab, vertical tab, line feed, carriage return and form feed characters.

ctype_upper (PHP 4 >= 4.0.4)

Check for uppercase character(s)

bool **ctype_upper** (string text) \linebreak

Returns `TRUE` if every character in *text* is a uppercase letter in the current locale.

See also `ctype_alpha()` and `ctype_lower()`.

ctype_xdigit (PHP 4 >= 4.0.4)

Check for character(s) representing a hexadecimal digit

bool **ctype_xdigit** (string text) \linebreak

Returns `TRUE` if every character in *text* is a hexadecimal 'digit', that is a decimal digit or a character from `[A-Fa-f]`, `FALSE` otherwise.

See also `ctype_digit()`.

XVII. Database (dbm-style) abstraction layer functions

Introduction

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 common subset of features supported by modern databases such as Sleepycat Software's DB2 (<http://www.sleepycat.com/>). (This is not to be confused with IBM's DB2 software, which is supported through the ODBC functions.)

Requirements

The behaviour of various aspects depends 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. You have to download and install supported dba-Handlers.

Table 1. List of DBA handlers

Handler	Notes
dbm	Dbm is the oldest (original) type of Berkeley DB style databases. You should avoid it, if possible. We do not support the compatibility functions built into DB2 and gdbm, because they are only compatible on the source code level, but cannot handle the original dbm format.
ndbm	Ndbm is a newer type and more flexible than dbm. It still has most of the arbitrary limits of dbm (therefore it is deprecated).
gdbm	Gdbm is the GNU database manager (ftp://ftp.gnu.org/pub/gnu/gdbm/).
db2	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.
db3	DB3 is Sleepycat Software's DB3 (http://www.sleepycat.com/).

Handler	Notes
cdb	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://cr.yip.to/cdb.html). Since it is constant, we support only reading operations.

When invoking the `dba_open()` or `dba_popen()` functions, one of the handler names must be supplied as an argument. The actually available list of handlers is displayed by invoking `phpinfo()`.

Installation

By using the `--enable-dba=shared` configuration option you can build a dynamic loadable module to enable PHP for basic support of dbm-style databases. You also have to add support for at least one of the following handlers by specifying the `--with-xxxx` configure switch to your PHP configure line.

Table 2. Supported DBA handlers

Handler	Configure Switch
dbm	To enable support for dbm add <code>--with-dbm[=DIR]</code> .
ndbm	To enable support for ndbm add <code>--with-ndbm[=DIR]</code> .
gdbm	To enable support for gdbm add <code>--with-gdbm[=DIR]</code> .
db2	To enable support for db2 add <code>--with-db2[=DIR]</code> .
db3	To enable support for db3 add <code>--with-db3[=DIR]</code> .
cdb	To enable support for cdb add <code>--with-cdb[=DIR]</code> .

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

The functions `dba_open()` and `dba_popen()` return a handle to the specified database file to access which is used by all other dba-function calls.

Predefined Constants

This extension does not define any constants.

Examples

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. However, 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 >= 4.0.0)

Close database

```
void dba_close ( resource handle) \linebreak
```

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 >= 4.0.0)

Delete entry specified by key

```
bool dba_delete ( string key, resource handle) \linebreak
```

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 >= 4.0.0)

Check whether key exists

```
bool dba_exists ( string key, resource handle) \linebreak
```

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 >= 4.0.0)

Fetch data specified by key

string **dba_fetch** (string *key*, resource *handle*) \linebreak

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 >= 4.0.0)

Fetch first key

string **dba_firstkey** (resource *handle*) \linebreak

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()` and example 2 in the DBA examples

dba_insert (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Insert entry

bool **dba_insert** (string *key*, string *value*, resource *handle*) \linebreak

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 >= 4.0.0)

Fetch next key

string **dba_nextkey** (resource handle) \linebreak

dba_nextkey() returns the next key of the database specified by *handle* and advances 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()` and example 2 in the DBA examples

dba_open (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Open database

resource **dba_open** (string path, string mode, string handler [, ...]) \linebreak

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 handle 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 >= 4.0.0)

Optimize database

bool **dba_optimize** (resource handle) \linebreak

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_popen (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Open database persistently

resource **dba_popen** (string path, string mode, string handler [, ...]) \linebreak

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 handle or `FALSE`, in the case the open is successful or fails, respectively.

See also: `dba_open()` `dba_close()`

dba_replace (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Replace or insert entry

bool **dba_replace** (string key, string value, resource handle) \linebreak

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 >= 4.0.0)

Synchronize database

bool **dba_sync** (resource handle) \linebreak

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()`

XVIII. Date and Time functions

Introduction

You can use these functions to handle date and time. These functions allow you to get date and time from the server where PHP is running on. You can use these functions to format the output of date and time in many different ways.

Note: Please keep in mind that these functions are depending on the locale settings of your server. Especially consider daylight saving time settings and leap years.

Requirements

These functions are available as part of the standard module, which is always available.

Installation

There is no installation needed to use these functions, they are part of the PHP core.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

This extension does not define any constants.

checkdate (PHP 3, PHP 4 >= 4.0.0)

Validate a gregorian date/time

bool **checkdate** (int month, int day, int year) \linebreak

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.

See also `mktime()` and `strtotime()`.

date (PHP 3, PHP 4 >= 4.0.0)

Format a local time/date

string **date** (string format [, int timestamp]) \linebreak

Returns a string formatted according to the given format string using the given integer *timestamp* or the current local time if no timestamp is given.

Note: The valid range of a timestamp is typically from Fri, 13 Dec 1901 20:45:54 GMT to Tue, 19 Jan 2038 03:14:07 GMT. (These are the dates that correspond to the minimum and maximum values for a 32-bit signed integer.)

To generate a timestamp from a string representation of the date, you may be able to use `strtotime()`. Additionally, some databases have functions to convert their date formats into timestamps (such as MySQL's `UNIX_TIMESTAMP` function).

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"
- O - Difference to Greenwich time in hours; i.e. "+0200"
- r - RFC 822 formatted date; i.e. "Thu, 21 Dec 2000 16:01:07 +0200" (added in PHP 4.0.4)
- s - seconds; i.e. "00" to "59"
- S - English ordinal suffix for the day of the month, 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)
- W - ISO-8601 week number of year, weeks starting on monday (added in PHP 4.1.0) (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 gmdate().

Example 1. date() example

```
echo date ("l dS of F Y h:i:s A");
echo "July 1, 2000 is on a " . date ("l", mktime(0,0,0,7,1,2000));
```

You can prevent a recognized character in the format string from being expanded by escaping it with a preceding backslash. If the character with a backslash is already a special sequence, you may need to also escape the backslash.

Example 2. Escaping characters in date()

```
echo date("l \\t\\h\\e jS"); // prints something like 'Saturday the 8th'
```

It is possible to use **date()** and **mktime()** together to find dates in the future or the past.

Example 3. 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);
```

Note: This can be more reliable than simply adding or subtracting the number of seconds in a day or month to a timestamp because of daylight savings time.

Some examples of **date()** formatting. Note that you should escape any other characters, as any which currently have a special meaning will produce undesirable results, and other characters may be assigned meaning in future PHP versions. When escaping, be sure to use single quotes to prevent characters like `\n` from becoming newlines.

Example 4. date() Formatting

```
/* Today is March 10th, 2001, 5:16:18 pm */
$today = date("F j, Y, g:i a"); // March 10, 2001, 5:16 pm
$today = date("m.d.y"); // 03.10.01
$today = date("j, n, Y"); // 10, 3, 2001
$today = date("Ymd"); // 20010310
$today = date('h-i-s, j-m-y, it is w Day z '); // 05-16-17, 10-03-01, 1631 1618 6 Fripm01
$today = date('\i\t \i\s \t\\h\\e jS \\d\\a\\y. '); // It is the 10th day.
$today = date("D M j G:i:s T Y"); // Sat Mar 10 15:16:08 MST 2001
$today = date('H:m:s \m \i\s\ \m\\o\\n\\t\\h'); // 17:03:17 m is month
$today = date("H:i:s"); // 17:16:17
```

To format dates in other languages, you should use the `setlocale()` and `strftime()` functions.

See also `getlastmod()`, `gmdate()`, `mktime()`, `strftime()` and `time()`.

getdate (PHP 3, PHP 4 >= 4.0.0)

Get date/time information

array **getdate** ([int timestamp]) \linebreak

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 : from 0 as Sunday up to 6 as Saturday
- "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"

Example 1. getdate() example

```
$today = getdate();
$month = $today['month'];
$mday = $today['mday'];
$year = $today['year'];
echo "$month $mday, $year";
```

gettimeofday (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Get current time

array **gettimeofday** (void) \linebreak

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 >= 4.0.0)

Format a GMT/CUT date/time

string **gmdate** (string format [, int timestamp]) \linebreak

Identical to the `date()` function except that the time returned is Greenwich Mean Time (GMT). For example, when run in Finland (GMT +0200), the first line below prints "Jan 01 1998 00:00:00", while the second prints "Dec 31 1997 22:00:00".

Example 1. gmdate() example

```
echo date ("M d Y H:i:s", mktime (0,0,0,1,1,1998));
echo gmdate ("M d Y H:i:s", mktime (0,0,0,1,1,1998));
```

See also `date()`, `mktime()`, `gmmktime()` and `strftime()`.

gmmktime (PHP 3, PHP 4 >= 4.0.0)

Get UNIX timestamp for a GMT date

int **gmmktime** (int hour, int minute, int second, int month, int day, int year [, int is_dst]) \linebreak

Identical to `mktime()` except the passed parameters represents a GMT date.

gmstrftime (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Format a GMT/CUT time/date according to locale settings

string **gmstrftime** (string format [, int timestamp]) \linebreak

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 >= 4.0.0)

Get the local time

array **localtime** ([int timestamp [, bool is_associative]]) \linebreak

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 then 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, starting with 0 for January
- "tm_year" - Years since 1900
- "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 >= 4.0.0)

Return current UNIX timestamp with microseconds

string **microtime** (void) \linebreak

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.

Both portions of the string are returned in units of seconds.

Example 1. microtime() example

```
function getmicrotime(){
    list($usec, $sec) = explode(" ",microtime());
    return ((float)$usec + (float)$sec);
}

$time_start = getmicrotime();

for ($i=0; $i < 1000; $i++){
    //do nothing, 1000 times
}

$time_end = getmicrotime();
$time = $time_end - $time_start;

echo "Did nothing in $time seconds";
```

See also time().

mktime

 (PHP 3, PHP 4 >= 4.0.0)

Get UNIX timestamp for a date

int **mktime** (int hour, int minute, int second, int month, int day, int year [, int is_dst]) \linebreak

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. If it's unknown, PHP tries to figure it out itself. This can cause unexpected (but not incorrect) results.

Note: *is_dst* was added in 3.0.10.

mktime() is useful for doing date arithmetic and validation, as it will automatically calculate the correct value for out-of-range input. For example, each of the following lines produces the string "Jan-01-1998".

Example 1. mktime() example

```
echo date ("M-d-Y", mktime (0,0,0,12,32,1997));
echo date ("M-d-Y", mktime (0,0,0,13,1,1997));
echo date ("M-d-Y", mktime (0,0,0,1,1,1998));
echo date ("M-d-Y", mktime (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 = mktime (0,0,0,3,0,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);

$lastday = mktime (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 what be regarded as 30.11.1999, which would be strange behavior).

See also `date()` and `time()`.

strftime (PHP 3, PHP 4 >= 4.0.0)

Format a local time/date according to locale settings

string **strftime** (string format [, int timestamp]) \linebreak

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')
- %g - like %G, but without the century.
- %G - The 4-digit year corresponding to the ISO week number (see %V). This has the same format and value as %Y, except that if the ISO week number belongs to the previous or next year, that year is used instead.
- %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

Warning

Sun Solaris seems to start with Sunday as 1 although ISO 9889:1999 (the current C standard) clearly specifies that it should be 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. (Use %G or %g for the year component that corresponds to the week number for the specified timestamp.)
- %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()**. This means that %T and %D will not work on Windows.

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_FR");
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 `strptime()` (<http://www.opengroup.org/onlinepubs/7908799/xsh/strptime.html>).

strtotime (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

Parse about any English textual datetime description into a UNIX timestamp

```
int strtotime ( string time [, int now]) \linebreak
```

The function expects to be given a string containing an English date format and will try to parse that format into a UNIX timestamp relative to the timestamp given in *now*, or the current time if none is supplied. Upon failure, `-1` is returned.

Because `strtotime()` behaves according to GNU date syntax, have a look at the GNU manual page titled Date Input Formats (http://www.gnu.org/manual/tar-1.12/html_chapter/tar_7.html). Described there is valid syntax for the *time* parameter.

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";
echo strtotime ("next Thursday"), "\n";
echo strtotime ("last Monday"), "\n";
```

Example 2. Checking for failure

```
$str = 'Not Good';
if (($timestamp = strtotime($str)) === -1) {
    echo "The string ($str) is bogus";
} else {
    echo "$str == ". date('l dS of F Y h:i:s A', $timestamp);
}
```

Note: The valid range of a timestamp is typically from Fri, 13 Dec 1901 20:45:54 GMT to Tue, 19 Jan 2038 03:14:07 GMT. (These are the dates that correspond to the minimum and maximum values for a 32-bit signed integer.)

time (PHP 3, PHP 4 >= 4.0.0)

Return current UNIX timestamp

int **time** (void) \linebreak

Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

See also date().

XIX. dBase functions

Introduction

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.

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, because the file format is commonly understood by Windows spreadsheets and organizers.

Installation

In order to use these functions, you must compile PHP with the `--enable-dbase` option.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

This extension does not define any resource types.

Predefined Constants

This extension does not define any constants.

dbase_add_record (PHP 3, PHP 4 >= 4.0.0)

Add a record to a dBase database

```
bool dbase_add_record ( int dbase_identifier, array record) \linebreak
```

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_close (PHP 3, PHP 4 >= 4.0.0)

Close a dBase database

```
bool dbase_close ( int dbase_identifier) \linebreak
```

Closes the database associated with *dbase_identifier*.

dbase_create (PHP 3, PHP 4 >= 4.0.0)

Creates a dBase database

```
int dbase_create ( string filename, array fields) \linebreak
```

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_delete_record (PHP 3, PHP 4 >= 4.0.0)

Deletes a record from a dBase database

```
bool dbase_delete_record ( int dbase_identifier, int record) \linebreak
```

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 >= 4.0.0)

Gets a record from a dBase database

```
array dbase_get_record ( int dbase_identifier, int record) \linebreak
```

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, except:

- Dates are left as strings
- Integers that would have caused an overflow (> 32 bits) are returned as strings

dbase_get_record_with_names (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Gets a record from a dBase database as an associative array

array **dbase_get_record_with_names** (int *dbase_identifier*, int *record*) \linebreak

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, except:

- Dates are left as strings
- Integers that would have caused an overflow (> 32 bits) are returned as strings

dbase_numfields (PHP 3, PHP 4 >= 4.0.0)

Find out how many fields are in a dBase database

int **dbase_numfields** (int *dbase_identifier*) \linebreak

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 >= 4.0.0)

Find out how many records are in a dBase database

int **dbase_numrecords** (int *dbase_identifier*) \linebreak

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`.

dbase__open (PHP 3, PHP 4 >= 4.0.0)

Opens a dBase database

`int dbase__open (string filename, int flags) \linebreak`

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.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

dbase__pack (PHP 3, PHP 4 >= 4.0.0)

Packs a dBase database

`bool dbase__pack (int dbase_identifier) \linebreak`

Packs the specified database (permanently deleting all records marked for deletion using `dbase_delete_record()`).

dbase__replace__record (PHP 3>= 3.0.11, PHP 4 >= 4.0.0)

Replace a record in a dBase database

`bool dbase__replace__record (int dbase_identifier, array record, int dbase_record_number) \linebreak`

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()`).

XX. 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);
```

dblist (PHP 3, PHP 4 >= 4.0.0)

Describes the DBM-compatible library being used

string **dblist** (void) \linebreak

dbmclose (PHP 3, PHP 4 >= 4.0.0)

Closes a dbm database

bool **dbmclose** (int dbm_identifier) \linebreak

Unlocks and closes the specified database.

dbmdelete (PHP 3, PHP 4 >= 4.0.0)

Deletes the value for a key from a DBM database

bool **dbmdelete** (int dbm_identifier, string key) \linebreak

Deletes the value for *key* in the database.

Returns `FALSE` if the key didn't exist in the database.

dbmexists (PHP 3, PHP 4 >= 4.0.0)

Tells if a value exists for a key in a DBM database

bool **dbmexists** (int dbm_identifier, string key) \linebreak

Returns `TRUE` if there is a value associated with the *key*.

dbmfetch (PHP 3, PHP 4 >= 4.0.0)

Fetches a value for a key from a DBM database

string **dbmfetch** (int dbm_identifier, string key) \linebreak

Returns the value associated with *key*.

dbmfirstkey (PHP 3, PHP 4 >= 4.0.0)

Retrieves the first key from a DBM database

string **dbmfirstkey** (int dbm_identifier) \linebreak

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.

dbminsert (PHP 3, PHP 4 >= 4.0.0)

Inserts a value for a key in a DBM database

int **dbminsert** (int dbm_identifier, string key, string value) \linebreak

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()`.)

dbmnextkey (PHP 3, PHP 4 >= 4.0.0)

Retrieves the next key from a DBM database

string **dbmnextkey** (int dbm_identifier, string key) \linebreak

Returns the next key after *key*. By calling `dbmfirstkey()` followed by successive calls to `dbmnextkey()` it is possible to visit every key/value pair in the dbm database. For example:

Example 1. Visiting every key/value pair in a DBM database

```
$key = dbmfirstkey ($dbm_id);
while ($key) {
    echo "$key = " . dbmfetch ($dbm_id, $key) . "\n";
    $key = dbmnextkey ($dbm_id, $key);
}
```

dbmopen (PHP 3, PHP 4 >= 4.0.0)

Opens a DBM database

int **dbmopen** (string filename, string flags) \linebreak

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 .lock 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/>).

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

dbmreplace (PHP 3, PHP 4 >= 4.0.0)

Replaces the value for a key in a DBM database

int **dbmreplace** (int dbm_identifier, string key, string value) \linebreak

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.

XXI. dbx functions

The dbx module is a database abstraction layer (db 'X', where 'X' is a supported database). The dbx functions allow you to access all supported databases using a single calling convention. In order to have these functions available, you must compile PHP with dbx support by using the `--enable-dbx` option and all options for the databases that will be used, e.g. for MySQL you must also specify `--with-mysql`. The dbx-functions themselves do not interface directly to the databases, but interface to the modules that are used to support these databases. To be able to use a database with the dbx-module, the module must be either linked or loaded into PHP, and the database module must be supported by the dbx-module. Currently, MySQL, PostgreSQL, Microsoft SQL Server, FrontBase, Sybase-CT and ODBC are supported, but others will follow (soon, I hope :-).

Documentation for adding additional database support to dbx can be found at <http://www.guidance.nl/php/dbx/doc/>.

dbx_close (PHP 4 >= 4.0.6)

Close an open connection/database

bool **dbx_close** (object link_identifier) \linebreak

Returns TRUE on success, FALSE on failure.

Example 1. dbx_close() example

```
<?php
$link = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password")
    or die ("Could not connect");

print("Connected successfully");
dbx_close($link);
?>
```

Note: Always refer to the module-specific documentation as well.

See also: dbx_connect().

dbx_compare (PHP 4 >= 4.1.0)

Compare two rows for sorting purposes

int **dbx_compare** (array row_a, array row_b, string column_key [, int flags]) \linebreak

dbx_compare() returns 0 if the `row_a[$column_key]` is equal to `row_b[$column_key]`, and 1 or -1 if the former is greater or is smaller than the latter one, respectively, or vice versa if the *flag* is set to `DBX_CMP_DESC`. **dbx_compare()** is a helper function for `dbx_sort()` to ease the make and use of the custom sorting function.

The *flags* can be set to specify comparison direction:

- `DBX_CMP_ASC` - ascending order
- `DBX_CMP_DESC` - descending order

and the preferred comparison type:

- `DBX_CMP_NATIVE` - no type conversion
- `DBX_CMP_TEXT` - compare items as strings
- `DBX_CMP_NUMBER` - compare items numerically

One of the direction and one of the type constant can be combined with bitwise OR operator (`|`). The default value for the *flags* parameter is `DBX_CMP_ASC | DBX_CMP_NATIVE`.

Example 1. dbx_compare() example

```
<?php
function user_re_order ($a, $b) {
    $rv = dbx_compare ($a, $b, "parentid", DBX_CMP_DESC);
    if ( !$rv ) {
        $rv = dbx_compare ($a, $b, "id", DBX_CMP_NUMBER);
    }
    return $rv;
}

$link = dbx_connect (DBX_ODBC, "", "db", "username", "password")
or die ("Could not connect");

$result = dbx_query ($link, "SELECT id, parentid, description FROM table ORDER BY id");
// data in $result is now ordered by id

dbx_sort ($result, "user_re_order");
// data in $result is now ordered by parentid (descending), then by id

dbx_close ($link);
?>
```

See also `dbx_sort()`.

dbx_connect (PHP 4 >= 4.0.6)

Open a connection/database

object **dbx_connect** (mixed module, string host, string database, string username, string password [, int persistent]) \linebreak

dbx_connect() returns an object on success, `FALSE` on error. If a connection has been made but the database could not be selected, the connection is closed and `FALSE` is returned. The *persistent* parameter can be set to `DBX_PERSISTENT`, if so, a persistent connection will be created.

The *module* parameter can be either a string or a constant, though the latter form is preferred. The possible values are given below, but keep in mind that they only work if the module is actually loaded.

- `DBX_MYSQL` or "mysql"
- `DBX_ODBC` or "odbc"
- `DBX_PGSQL` or "pgsql"

- DBX_MSSQL or "mssql"
- DBX_FBSQL or "fbsql" (available from PHP 4.1.0)
- DBX_SYBASECT or "sybase_ct" (CVS only)

The *host*, *database*, *username* and *password* parameters are expected, but not always used depending on the connect functions for the abstracted module.

The returned object has three properties:

database

It is the name of the currently selected database.

handle

It is a valid handle for the connected database, and as such it can be used in module-specific functions (if required).

```
$link = dbx_connect (DBX_MYSQL, "localhost", "db", "username", "password");
mysql_close ($link->handle); // dbx_close($link) would be better here
```

module

It is used internally by dbx only, and is actually the module number mentioned above.

Example 1. dbx_connect() example

```
<?php
$link = dbx_connect (DBX_ODBC, "", "db", "username", "password", DBX_PERSISTENT)
    or die ("Could not connect");

print ("Connected successfully");
dbx_close ($link);
?>
```

Note: Always refer to the module-specific documentation as well.

See also: `dbx_close()`.

dbx_error (PHP 4 >= 4.0.6)

Report the error message of the latest function call in the module (not just in the connection)

string **dbx_error** (object link_identifier) \linebreak

dbx_error() returns a string containing the error message from the last function call of the abstracted module (e.g. mysql module). If there are multiple connections in the same module, just the last error is given. If there are connections on different modules, the latest error is returned for the module specified by the *link_identifier* parameter.

Example 1. dbx_error() example

```
<?php
$link = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password")
      or die ("Could not connect");

$result = dbx_query($link, "select id from non_existing_table");
if ( $result == 0 ) {
    echo dbx_error ($link);
}
dbx_close ($link);
?>
```

Note: Always refer to the module-specific documentation as well.

The error message for Microsoft SQL Server is actually the result of the `mssql_get_last_message()` function.

dbx_query (PHP 4 >= 4.0.6)

Send a query and fetch all results (if any)

object **dbx_query** (object link_identifier, string sql_statement [, long flags]) \linebreak

dbx_query() returns an object or 1 on success, and 0 on failure. The result object is returned only if the query given in *sql_statement* produces a result set.

Example 1. How to handle the returned value

```
<?php
$link = dbx_connect(DBX_ODBC, "", "db", "username", "password")
      or die("Could not connect");
```

```

$result = dbx_query($link, 'SELECT id, parentid, description FROM table');

if ( is_object($result) ) {
    // ... do some stuff here, see detailed examples below ...
    // first, print out field names and types
    // then, draw a table filled with the returned field values
}
else if ( $result == 1 ) {
    echo("Query executed successfully, but no result set returned");
}
else {
    exit("Query failed");
}

dbx_close($link);
?>

```

The *flags* parameter is used to control the amount of information that is returned. It may be any combination of the following constants with the bitwise OR operator (`|`):

DBX_RESULT_INDEX

It is *always* set, that is, the returned object has a `data` property which is a 2 dimensional array indexed numerically. For example, in the expression `data[2][3]` 2 stands for the row (or record) number and 3 stands for the column (or field) number. The first row and column are indexed at 0.

If `DBX_RESULT_ASSOC` is also specified, the returning object contains the information related to `DBX_RESULT_INFO` too, even if it was not specified.

DBX_RESULT_INFO

It provides info about columns, such as field names and field types.

DBX_RESULT_ASSOC

It effects that the field values can be accessed with the respective column names used as keys to the returned object's `data` property.

Associated results are actually references to the numerically indexed data, so modifying `data[0][0]` causes that `data[0]['field_name_for_first_column']` is modified as well.

Note that `DBX_RESULT_INDEX` is always used, regardless of the actual value of *flags* parameter. This means that the following combinations is effective only:

- `DBX_RESULT_INDEX`
- `DBX_RESULT_INDEX | DBX_RESULT_INFO`

- DBX_RESULT_INDEX | DBX_RESULT_INFO | DBX_RESULT_ASSOC - this is the default, if *flags* is not specified.

The returning object has four or five properties depending on *flags*:

handle

It is a valid handle for the connected database, and as such it can be used in module specific functions (if required).

```
$result = dbx_query ($link, "SELECT id FROM table");
mysql_field_len ($result->handle, 0);
```

cols and rows

These contain the number of columns (or fields) and rows (or records) respectively.

```
$result = dbx_query ($link, 'SELECT id FROM table');
echo $result->rows; // number of records
echo $result->cols; // number of fields
```

info (optional)

It is returned only if either DBX_RESULT_INFO or DBX_RESULT_ASSOC is specified in the *flags* parameter. It is a 2 dimensional array, that has two named rows (name and type) to retrieve column information.

Example 2. lists each field's name and type

```
$result = dbx_query ($link, 'SELECT id FROM table',
                    DBX_RESULT_INDEX | DBX_RESULT_INFO);

for ($i = 0; $i < $result->cols; $i++ ) {
    echo $result->info['name'][$i] . "\n";
    echo $result->info['type'][$i] . "\n";
}
```


data

This property contains the actual resulting data, possibly associated with column names as well depending on *flags*. If `DBX_RESULT_ASSOC` is set, it is possible to use `$result->data[2]["field_name"]`.

Example 3. outputs the content of data property into HTML table

```
$result = dbx_query ($link, 'SELECT id, parentid, description FROM table');

echo "<table>\n";
foreach ( $result->data as $row ) {
    echo "<tr>\n";
    foreach ( $row as $field ) {
        echo "<td>$field</td>";
    }
    echo "</tr>\n";
}
echo "</table>\n";
```

Note: Always refer to the module-specific documentation as well.

See also: `dbx_connect()`.

dbx_sort (PHP 4 >= 4.0.6)

Sort a result from a `dbx_query` by a custom sort function

bool **dbx_sort** (object result, string user_compare_function) \linebreak

Returns `TRUE` on success, `FALSE` on failure.

Note: It is always better to use `ORDER BY SQL` clause instead of **dbx_sort()**, if possible.

Example 1. dbx_sort() example

```
<?php
function user_re_order ($a, $b) {
    $rv = dbx_compare ($a, $b, "parentid", DBX_CMP_DESC);
```

```
    if ( !$rv ) {
        $rv = dbx_compare ($a, $b, "id", DBX_CMP_NUMBER);
    }
    return $rv;
}

$link = dbx_connect (DBX_ODBC, "", "db", "username", "password")
    or die ("Could not connect");

$result = dbx_query ($link, "SELECT id, parentid, description FROM tbl ORDER BY id");
    // data in $result is now ordered by id

dbx_sort ($result, "user_re_order");
    // data in $result is now ordered by parentid (descending), then by id

dbx_close ($link);
?>
```

See also `dbx_compare()`.

XXII. DB++ Functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

Introduction

db++, made by the german company Concept asa (<http://www.concept-asa.de/>), is a relational database system with high performance and low memory and disk usage in mind. While providing SQL as an additional language interface it is not really a SQL database in the first place but provides its own AQL query language which is much more influenced by the relational algebra then SQL is.

Concept asa always had an interest in supporting open source languages, db++ has had Perl and Tcl call interfaces for years now and uses Tcl as its internal stored procedure language.

Requirements

This extension relies on external client libraries so you have to have a db++ client installed on the system you want to use this extension on.

Concept asa (<http://www.concept-asa.de/>) provides db++ Demo versions (<http://www.concept-asa.de/down-eng.html>) and documentation (<http://www.concept-asa.de/downloads/doc-eng.tar.gz>) for Linux, some other UNIX versions. There is also a Windows version of db++, but this extension doesn't support it (yet).

Installation

In order to build this extension yourself you need the db++ client libraries and header files to be installed on your system (these are included in the db++ installation archives by default). You have to run **configure** with option `--with-dbplus` to build this extension.

configure looks for the client libraries and header files under the default paths `/usr/dbplus`, `/usr/local/dbplus` and `/opt/dbplus`. If you have installed db++ in a different place you have add

the installation path to the **configure** option like this: `--with-dbplus=/your/installation/path`.

Runtime Configuration

This extension does not define any configuration directives.

Resource Types

dbplus_relation

Most db++ functions operate on or return *dbplus_relation* resources. A *dbplus_relation* is a handle to a stored relation or a relation generated as the result of a query.

Predefined Constants

These constants are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

db++ error codes

Table 1. DB++ Error Codes

PHP Constant	db++ constant	meaning
DBPLUS_ERR_NOERR (integer)	ERR_NOERR	Null error condition
DBPLUS_ERR_DUPLICATE (integer)	ERR_DUPLICATE	Tried to insert a duplicate tuple
DBPLUS_ERR_EOSCAN (integer)	ERR_EOSCAN	End of scan from rget()
DBPLUS_ERR_EMPTY (integer)	ERR_EMPTY	Relation is empty (server)
DBPLUS_ERR_CLOSE (integer)	ERR_CLOSE	The server can't close
DBPLUS_ERR_WLOCKED (integer)	ERR_WLOCKED	The record is write locked
DBPLUS_ERR_LOCKED (integer)	ERR_LOCKED	Relation was already locked
DBPLUS_ERR_NOLOCK (integer)	ERR_NOLOCK	Relation cannot be locked
DBPLUS_ERR_READ (integer)	ERR_READ	Read error on relation
DBPLUS_ERR_WRITE (integer)	ERR_WRITE	Write error on relation
DBPLUS_ERR_CREATE (integer)	ERR_CREATE	Create() system call failed
DBPLUS_ERR_LSEEK (integer)	ERR_LSEEK	Lseek() system call failed

PHP Constant	db++ constant	meaning
DBPLUS_ERR_LENGTH (integer)	ERR_LENGTH	Tuple exceeds maximum length
DBPLUS_ERR_OPEN (integer)	ERR_OPEN	Open() system call failed
DBPLUS_ERR_WOPEN (integer)	ERR_WOPEN	Relation already opened for writing
DBPLUS_ERR_MAGIC (integer)	ERR_MAGIC	File is not a relation
DBPLUS_ERR_VERSION (integer)	ERR_VERSION	File is a very old relation
DBPLUS_ERR_PGFSIZE (integer)	ERR_PGFSIZE	Relation uses a different page size
DBPLUS_ERR_CRC (integer)	ERR_CRC	Invalid crc in the superpage
DBPLUS_ERR_PIPE (integer)	ERR_PIPE	Piped relation requires lseek()
DBPLUS_ERR_NIDX (integer)	ERR_NIDX	Too many secondary indices
DBPLUS_ERR_MALLOC (integer)	ERR_MALLOC	Malloc() call failed
DBPLUS_ERR_NUSERS (integer)	ERR_NUSERS	Error use of max users
DBPLUS_ERR_PREEXIT (integer)	ERR_PREEXIT	Caused by invalid usage
DBPLUS_ERR_ONTRAP (integer)	ERR_ONTRAP	Caused by a signal
DBPLUS_ERR_PREPROC (integer)	ERR_PREPROC	Error in the preprocessor
DBPLUS_ERR_DBPARSE (integer)	ERR_DBPARSE	Error in the parser
DBPLUS_ERR_DBRUNERR (integer)	ERR_DBRUNERR	Run error in db
DBPLUS_ERR_DBPREEXIT (integer)	ERR_DBPREEXIT	Exit condition caused by prexit() * procedure
DBPLUS_ERR_WAIT (integer)	ERR_WAIT	Wait a little (Simple only)
DBPLUS_ERR_CORRUPT_TUPLE (integer)	ERR_CORRUPT_TUPLE	A client sent a corrupt tuple
DBPLUS_ERR_WARNING0 (integer)	ERR_WARNING0	The Simple routines encountered a non fatal error which was corrected
DBPLUS_ERR_PANIC (integer)	ERR_PANIC	The server should not really die but after a disaster send ERR_PANIC to all its clients
DBPLUS_ERR_FIFO (integer)	ERR_FIFO	Can't create a fifo
DBPLUS_ERR_PERM (integer)	ERR_PERM	Permission denied
DBPLUS_ERR_TCL (integer)	ERR_TCL	TCL_error
DBPLUS_ERR_RESTRICTED (integer)	ERR_RESTRICTED	Only two users
DBPLUS_ERR_USER (integer)	ERR_USER	An error in the use of the library by an application programmer

PHP Constant	db++ constant	meaning
DBPLUS_ERR_UNKNOWN (integer)	ERR_UNKNOWN	

dbplus_add (PHP 4 >= 4.1.0)

Add a tuple to a relation

int **dbplus_add** (resource relation, array tuple) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

This function will add a tuple to a relation. The *tuple* data is an array of attribute/value pairs to be inserted into the given *relation*. After successful execution the *tuple* array will contain the complete data of the newly created tuple, including all implicitly set domain fields like sequences.

The function will return zero (aka. DBPLUS_ERR_NOERR) on success or a db++ error code on failure. See `dbplus_errcode()` or the introduction to this chapter for more information on db++ error codes.

dbplus_aql (PHP 4 >= 4.1.0)

Perform AQL query

resource **dbplus_aql** (string query [, string server [, string dbpath]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_aql() will execute an AQL *query* on the given *server* and *dbpath*.

On success it will return a relation handle. The result data may be fetched from this relation by calling `dbplus_next()` and **dbplus_current()**. Other relation access functions will not work on a result relation.

Further information on the AQL A... Query Language is provided in the original db++ manual.

dbplus_chdir (PHP 4 >= 4.1.0)

Get/Set database virtual current directory

string **dbplus_chdir** ([string newdir]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_chdir() will change the virtual current directory where relation files will be looked for by **dbplus_open()**. **dbplus_chdir()** will return the absolute path of the current directory. Calling **dbplus_chdir()** without giving any *newdir* may be used to query the current working directory.

dbplus_close (PHP 4 >= 4.1.0)

Close a relation

```
int dbplus_close ( resource relation) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Calling **dbplus_close()** will close a relation previously opened by **dbplus_open()**.

dbplus_curr (PHP 4 >= 4.1.0)

Get current tuple from relation

```
int dbplus_curr ( resource relation, array tuple) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_curr() will read the data for the current tuple for the given *relation* and will pass it back as an associative array in *tuple*.

The function will return zero (aka. `DBPLUS_ERR_NOERR`) on success or a db++ error code on failure. See **dbplus_errcode()** or the introduction to this chapter for more information on db++ error codes.

See also **dbplus_first()**, **dbplus_prev()**, **dbplus_next()**, and **dbplus_last()**.

dbplus_errcode (PHP 4 >= 4.1.0)

Get error string for given errorcode or last error

```
string dbplus_errcode ( int errno) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_errcode() returns a cleartext error string for the error code passed as *errno* or for the result code of the last db++ operation if no parameter is given.

dbplus_errno (PHP 4 >= 4.1.0)

Get error code for last operation

```
int dbplus_errno ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_errno() will return the error code returned by the last db++ operation.

See also `dbplus_errcode()`.

dbplus_find (PHP 4 >= 4.1.0)

Set a constraint on a relation

```
int dbplus_find ( resource relation, array constraints, mixed tuple) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_find() will place a constraint on the given relation. Further calls to functions like `dbplus_curr()` or `dbplus_next()` will only return tuples matching the given constraints.

Constraints are triplets of strings containing of a domain name, a comparison operator and a comparison value. The *constraints* parameter array may consist of a collection of string arrays, each of which contains a domain, an operator and a value, or of a single string array containing a multiple of three elements.

The comparison operator may be one of the following strings: '==', '>', '>=', '<', '<=', '!=', '~' for a regular expression match and 'BAND' or 'BOR' for bitwise operations.

See also `dbplus_unselect()`.

dbplus_first (PHP 4 >= 4.1.0)

Get first tuple from relation

```
int dbplus_first ( resource relation, array tuple) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`dbplus_curr()` will read the data for the first tuple for the given *relation*, make it the current tuple and pass it back as an associative array in *tuple*.

The function will return zero (aka. `DBPLUS_ERR_NOERR`) on success or a db++ error code on failure. See `dbplus_errcode()` or the introduction to this chapter for more information on db++ error codes.

See also `dbplus_curr()`, `dbplus_prev()`, `dbplus_next()`, and `dbplus_last()`.

dbplus_flush (PHP 4 >= 4.1.0)

Flush all changes made on a relation

```
int dbplus_flush ( resource relation) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`dbplus_flush()` will write all changes applied to *relation* since the last flush to disk.

The function will return zero (aka. `DBPLUS_ERR_NOERR`) on success or a db++ error code on failure. See `dbplus_errcode()` or the introduction to this chapter for more information on db++ error codes.

dbplus_freealllocks (PHP 4 >= 4.1.0)

Free all locks held by this client

```
int dbplus_freealllocks ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_freealllocks() will free all tuple locks held by this client.

See also `dbplus_getlock()`, `dbplus_freelock()`, and `dbplus_freerlocks()`.

dbplus_freelock (PHP 4 >= 4.1.0)

Release write lock on tuple

```
int dbplus_freelock ( resource relation, string tname) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_freelock() will release a write lock on the given *tuple* previously obtained by `dbplus_getlock()`.

See also `dbplus_getlock()`, `dbplus_freerlocks()`, and `dbplus_freealllocks()`.

dbplus_freerlocks (PHP 4 >= 4.1.0)

Free all tuple locks on given relation

```
int dbplus_freerlocks ( resource relation) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_freerlocks() will free all tuple locks held on the given *relation*.

See also `dbplus_getlock()`, `dbplus_freelock()`, and `dbplus_freealllocks()`.

dbplus_getlock (PHP 4 >= 4.1.0)

Get a write lock on a tuple

```
int dbplus_getlock ( resource relation, string tname ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_getlock() will request a write lock on the specified *tuple*. It will return zero on success or a non-zero error code, especially `DBPLUS_ERR_WLOCKED`, on failure.

See also `dbplus_freelock()`, `dbplus_freerlocks()`, and `dbplus_freealllocks()`.

dbplus_getunique (PHP 4 >= 4.1.0)

Get a id number unique to a relation

```
int dbplus_getunique ( resource relation, int uniqueid ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_getunique() will obtain a number guaranteed to be unique for the given *relation* and will pass it back in the variable given as *uniqueid*.

The function will return zero (aka. `DBPLUS_ERR_NOERR`) on success or a db++ error code on failure. See `dbplus_errcode()` or the introduction to this chapter for more information on db++ error codes.

dbplus_info (PHP 4 >= 4.1.0)

???

```
int dbplus_info ( resource relation, string key, array ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_last (PHP 4 >= 4.1.0)

Get last tuple from relation

```
int dbplus_last ( resource relation, array tuple) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_curr() will read the data for the last tuple for the given *relation*, make it the current tuple and pass it back as an associative array in *tuple*.

The function will return zero (aka. DBPLUS_ERR_NOERR) on success or a db++ error code on failure. See dbplus_errcode() or the introduction to this chapter for more information on db++ error codes.

See also dbplus_first(), dbplus_curr(), dbplus_prev(), and dbplus_next().

dbplus_lockrel (unknown)

Request write lock on relation

```
int dbplus_lockrel ( resource relation) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_lockrel() will request a write lock on the given relation. Other clients may still query the relation, but can't alter it while it is locked.

dbplus_next (PHP 4 >= 4.1.0)

Get next tuple from relation

```
int dbplus_next ( resource relation, array ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_curr() will read the data for the next tuple for the given *relation*, will make it the current tuple and will pass it back as an associative array in *tuple*.

The function will return zero (aka. DBPLUS_ERR_NOERR) on success or a db++ error code on failure. See dbplus_errcode() or the introduction to this chapter for more information on db++ error codes.

See also dbplus_first(), dbplus_curr(), dbplus_prev(), and dbplus_last().

dbplus_open (PHP 4 >= 4.1.0)

Open relation file

```
resource dbplus_open ( string name) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The relation file *name* will be opened. *name* can be either a file name or a relative or absolute path name. This will be mapped in any case to an absolute relation file path on a specific host machine and server.

On success a relation file resource (cursor) is returned which must be used in any subsequent commands referencing the relation. Failure leads to a zero return value, the actual error code may be asked for by calling dbplus_errno().

dbplus_prev (PHP 4 >= 4.1.0)

Get previous tuple from relation

```
int dbplus_prev ( resource relation, array tuple) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`dbplus_curr()` will read the data for the next tuple for the given *relation*, will make it the current tuple and will pass it back as an associative array in *tuple*.

The function will return zero (aka. `DBPLUS_ERR_NOERR`) on success or a db++ error code on failure. See `dbplus_errcode()` or the introduction to this chapter for more information on db++ error codes.

See also `dbplus_first()`, `dbplus_curr()`, `dbplus_next()`, and `dbplus_last()`.

dbplus_rchperm (PHP 4 >= 4.1.0)

Change relation permissions

```
int dbplus_rchperm ( resource relation, int mask, string user, string group) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`dbplus_rchperm()` will change access permissions as specified by *mask*, *user* and *group*. The values for these are operating system specific.

dbplus_rcreate (PHP 4 >= 4.1.0)

Creates a new DB++ relation

```
resource dbplus_rcreate ( string name, mixed domlist [, boolean overwrite]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`dbplus_rcreate()` will create a new relation named *name*. An existing relation by the same name will only be overwritten if the relation is currently not in use and *overwrite* is set to `TRUE`.

domlist should contain the domain specification for the new relation within an array of domain description strings. (`dbplus_rcreate()`) will also accept a string with space delimited domain description

strings, but it is recommended to use an array). A domain description string consists of a domain name unique to this relation, a slash and a type specification character. See the db++ documentation, especially the dbcreate(1) manpage, for a description of available type specifiers and their meanings.

dbplus_rcrtextact (PHP 4 >= 4.1.0)

Creates an exact but empty copy of a relation including indices

resource **dbplus_rcrtextact** (string name, resource relation, boolean overwrite) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rcrtextact() will create an exact but empty copy of the given *relation* under a new *name*. An existing relation by the same *name* will only be overwritten if *overwrite* is TRUE and no other process is currently using the relation.

dbplus_rcrtlike (PHP 4 >= 4.1.0)

Creates an empty copy of a relation with default indices

resource **dbplus_rcrtlike** (string name, resource relation, int flag) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rcrtlike() will create an empty copy of the given *relation* under a new *name*, but with default indices. An existing relation by the same *name* will only be overwritten if *overwrite* is TRUE and no other process is currently using the relation.

dbplus_resolve (PHP 4 >= 4.1.0)

Resolve host information for relation

int **dbplus_resolve** (string relation_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_resolve() will try to resolve the given *relation_name* and find out internal server id, real hostname and the database path on this host. The function will return an array containing these values under the keys 'sid', 'host' and 'host_path' or `FALSE` on error.

See also `dbplus_tcl()`.

dbplus_restorepos (PHP 4 >= 4.1.0)

???

int **dbplus_restorepos** (resource relation, array tuple) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_rkeys (PHP 4 >= 4.1.0)

Specify new primary key for a relation

resource **dbplus_rkeys** (resource relation, mixed domlist) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rkeys() will replace the current primary key for *relation* with the combination of domains specified by *domlist*.

domlist may be passed as a single domain name string or as an array of domain names.

dbplus_ropen (PHP 4 >= 4.1.0)

Open relation file local

resource **dbplus_ropen** (string name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_ropen() will open the relation *file* locally for quick access without any client/server overhead. Access is read only and only **dbplus_current()** and **dbplus_next()** may be applied to the returned relation.

dbplus_rquery (PHP 4 >= 4.1.0)

Perform local (raw) AQL query

int **dbplus_rquery** (string query, string dbpath) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rquery() performs a local (raw) AQL query using an AQL interpreter embedded into the db++ client library. **dbplus_rquery()** is faster than **dbplus_aql()** but will work on local data only.

dbplus_rename (PHP 4 >= 4.1.0)

Rename a relation

int **dbplus_rename** (resource relation, string name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rename() will change the name of *relation* to *name*.

dbplus_rsecindex (PHP 4 >= 4.1.0)

Create a new secondary index for a relation

resource **dbplus_rsecindex** (resource relation, mixed domlist, int type) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rsecindex() will create a new secondary index for *relation* with consists of the domains specified by *domlist* and is of type *type*

domlist may be passed as a single domain name string or as an array of domain names.

dbplus_runlink (PHP 4 >= 4.1.0)

Remove relation from filesystem

int **dbplus_runlink** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_unlink() will close and remove the *relation*.

dbplus_rzap (PHP 4 >= 4.1.0)

Remove all tuples from relation

int **dbplus_rzap** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_rzap() will remove all tuples from *relation*.

dbplus_savepos (PHP 4 >= 4.1.0)

???

int **dbplus_savepos** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_setindex (PHP 4 >= 4.1.0)

???

int **dbplus_setindex** (resource relation, string idx_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_setindexbynumber (PHP 4 >= 4.1.0)

???

int **dbplus_setindexbynumber** (resource relation, int idx_number) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_sql (PHP 4 >= 4.1.0)

Perform SQL query

resource **dbplus_sql** (string query, string server, string dbpath) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_tcl (PHP 4 >= 4.1.0)

Execute TCL code on server side

int **dbplus_tcl** (int sid, string script) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

A db++ server will prepare a TCL interpreter for each client connection. This interpreter will enable the server to execute TCL code provided by the client as a sort of stored procedures to improve the performance of database operations by avoiding client/server data transfers and context switches.

dbplus_tcl() needs to pass the client connection id the TCL *script* code should be executed by. **dbplus_resolve()** will provide this connection id. The function will return whatever the TCL code returns or a TCL error message if the TCL code fails.

See also **dbplus_resolve()**.

dbplus_tremove (PHP 4 >= 4.1.0)

Remove tuple and return new current tuple

int **dbplus_tremove** (resource relation, array tuple [, array current]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_tremove() removes *tuple* from *relation* if it perfectly matches a tuple within the relation. *current*, if given, will contain the data of the new current tuple after calling **dbplus_tremove()**.

dbplus_undo (PHP 4 >= 4.1.0)

???

int **dbplus_undo** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_undoprepere (PHP 4 >= 4.1.0)

???

int **dbplus_undoprepere** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Not implemented yet.

dbplus_unlockrel (PHP 4 >= 4.1.0)

Give up write lock on relation

int **dbplus_unlockrel** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_unlockrel() will release a write lock previously obtained by **dbplus_lockrel()**.

dbplus_unselect (PHP 4 >= 4.1.0)

Remove a constraint from relation

int **dbplus_unselect** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Calling **dbplus_unselect()** will remove a constraint previously set by **dbplus_find()** on *relation*.

dbplus_update (PHP 4 >= 4.1.0)

Update specified tuple in relation

int **dbplus_update** (resource relation, array old, array new) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_update() replaces the tuple given by *old* with the data from *new* if and only if *old* completely matches a tuple within *relation*.

dbplus_xlockrel (PHP 4 >= 4.1.0)

Request exclusive lock on relation

int **dbplus_xlockrel** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_xlockrel() will request an exclusive lock on *relation* preventing even read access from other clients.

See also `dbplus_xunlockrel()`.

dbplus_xunlockrel (PHP 4 >= 4.1.0)

Free exclusive lock on relation

int **dbplus_xunlockrel** (resource relation) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

dbplus_xunlockrel() will release an exclusive lock on *relation* previously obtained by `dbplus_xlockrel()`.

XXIII. Direct IO functions

Direct I/O Functions

PHP supports the direct io functions as described in the Posix Standard (Section 6) for performing I/O functions at a lower level than the C-Language stream I/O functions (fopen, fread,..).

Installation

To get these functions to work, you have to configure PHP with `--enable-dio`.

dio_close (PHP 4 >= 4.2.0)

Closes the file descriptor given by *fd*

```
void dio_close ( resource fd) \linebreak
```

The function **dio_close()** closes the file descriptor *resource*.

dio_fcntl (PHP 4 >= 4.2.0)

Performs a c library fcntl on *fd*

```
mixed dio_fcntl ( resource fd, int cmd [, mixed arg]) \linebreak
```

The **dio_fcntl()** function performs the operation specified by *cmd* on the file descriptor *fd*. Some commands require additional arguments *args* to be supplied.

arg is an associative array, when *cmd* is F_SETLK or F_SETLLW, with the following keys:

- "start" - offset where lock begins
- "length" - size of locked area. zero means to end of file
- "whence" - Where l_start is relative to: can be SEEK_SET, SEEK_END and SEEK_CUR
- "type" - type of lock: can be F_RDLCK (read lock), F_WRLCK (write lock) or F_UNLCK (unlock)

cmd can be one of the following operations:

- F_SETLK - Lock is set or cleared. If the lock is held by someone else **dio_fcntl()** returns -1.
- F_SETLKW - like F_SETLK, but in case the lock is held by someone else, **dio_fcntl()** waits until the lock is released.
- F_GETLK - **dio_fcntl()** returns an associative array (as described above) if someone else prevents lock. If there is no obstruction key "type" will set to F_UNLCK.
- F_DUPFD - finds the lowest numbered available file descriptor greater or equal than *arg* and returns them.

dio_open (PHP 4 >= 4.2.0)

Opens a new filename with specified permissions of flags and creation permissions of mode

```
resource dio_open ( string filename, int flags [, int mode]) \linebreak
```

dio_open() opens a file and returns a new file descriptor for it, or -1 if any error occurred. If *flags* is O_CREAT, optional third parameter *mode* will set the mode of the file (creation permissions). The *flags* parameter can be one of the following options:

- O_RDONLY - opens the file for read access
- O_WRONLY - opens the file for write access
- O_RDWR - opens the file for both reading and writing

The *flags* parameter can also include any combination of the following flags:

- O_CREAT - creates the file, if it doesn't already exist
- O_EXCL - if both, O_CREAT and O_EXCL are set, **dio_open()** fails, if file already exists
- O_TRUNC - if file exists, and its opened for write access, file will be truncated to zero length.
- O_APPEND - write operations write data at the end of file
- O_NONBLOCK - sets non blocking mode

dio_read (PHP 4 >= 4.2.0)

Reads *n* bytes from *fd* and returns them, if *n* is not specified, reads 1k block

string **dio_read** (resource *fd* [, int *n*]) \linebreak

The function **dio_read()** reads and returns *n* bytes from file with descriptor *resource*. If *n* is not specified, **dio_read()** reads 1K sized block and returns them.

dio_seek (PHP 4 >= 4.2.0)

Seeks to *pos* on *fd* from whence

int **dio_seek** (resource *fd*, int *pos*, int *whence*) \linebreak

The function **dio_seek()** is used to change the file position of the file with descriptor *resource*. The parameter *whence* specifies how the position *pos* should be interpreted:

- SEEK_SET - specifies that *pos* is specified from the beginning of the file
- SEEK_CUR - Specifies that *pos* is a count of characters from the current file position. This count may be positive or negative
- SEEK_END - Specifies that *pos* is a count of characters from the end of the file. A negative count specifies a position within the current extent of the file; a positive count specifies a position past the current end. If you set the position past the current end, and actually write data, you will extend the file with zeros up to that position

dio_stat (PHP 4 >= 4.2.0)

Gets stat information about the file descriptor `fd`

array **dio_stat** (resource `fd`) \linebreak

Function **dio_stat()** returns information about the file with file descriptor `fd`. **dio_stat()** returns an associative array with the following keys:

- "device" - device
- "inode" - inode
- "mode" - mode
- "nlink" - number of hard links
- "uid" - user id
- "gid" - group id
- "device_type" - device type (if inode device)
- "size" - total size in bytes
- "blocksize" - blocksize
- "blocks" - number of blocks allocated
- "atime" - time of last access
- "mtime" - time of last modification
- "ctime" - time of last change

On error **dio_stat()** returns NULL.

dio_truncate (PHP 4 >= 4.2.0)

Truncates file descriptor `fd` to offset bytes

bool **dio_truncate** (resource `fd`, int `offset`) \linebreak

Function **dio_truncate()** causes the file referenced by `fd` to be truncated to at most `offset` bytes in size. If the file previously was larger than this size, the extra data is lost. If the file previously was shorter, it is unspecified whether the file is left unchanged or is extended. In the latter case the extended part reads as zero bytes. Returns 0 on success, otherwise -1.

dio_write (PHP 4 >= 4.2.0)

Writes data to `fd` with optional truncation at length

int **dio_write** (resource `fd`, string `data` [, int `len`]) \linebreak

The function **dio_write()** writes up to *len* bytes from *data* to file *fd*. If *len* is not specified, **dio_write()** writes all *data* to the specified file. **dio_write()** returns the number of bytes written to *fd*.

XXIV. Directory functions

For related functions such as `dirname()`, `is_dir()`, `mkdir()`, and `rmdir()`, see the Filesystem section.

chdir (PHP 3, PHP 4 >= 4.0.0)

change directory

bool **chdir** (string directory) \linebreak

Changes PHP's current directory to *directory*. Returns `FALSE` if unable to change directory, `TRUE` otherwise.

chroot (PHP 4 >= 4.0.5)

change the root directory

bool **chroot** (string directory) \linebreak

Changes the root directory of the current process to *directory*. Returns `FALSE` if unable to change the root directory, `TRUE` otherwise.

Note: It's not wise to use this function when running in a webserver environment, because it's not possible to reset the root directory to `/` again at the end of the request. This function will only function correct when running as CGI this way.

dir (PHP 3, PHP 4 >= 4.0.0)

directory class

```
class dir {
    dir(string directory);
    string path;
    string read();
    void rewind();
    void close();
}
```

A pseudo-object oriented mechanism for reading a directory. The given *directory* is opened. Two properties are available once the 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`.

Please note the fashion in which **dir()**'s return value is checked in the example below. We are explicitly testing whether the return value is identical to (equal to and of the same type as--see [Comparison Operators](#) for more information) `FALSE` since otherwise, any directory entry whose name evaluates to `FALSE` will stop the loop.

Example 1. dir() example

```

$d = dir("/etc");
echo "Handle: " . $d->handle . "<br>\n";
echo "Path: " . $d->path . "<br>\n";
while (false !== ($entry = $d->read())) {
    echo $entry . "<br>\n";
}
$d->close();

```

Note: The order in which directory entries are returned by the read method is system-dependent.

Note: This defines the internal class `Directory`, meaning that you will not be able to define your own classes with that name. For a full list of predefined classes in PHP, please see Predefined Classes.

closedir (PHP 3, PHP 4 >= 4.0.0)

close directory handle

void **closedir** (resource *dir_handle*) \linebreak

Closes the directory stream indicated by *dir_handle*. The stream must have previously been opened by `opendir()`.

getcwd (PHP 4 >= 4.0.0)

gets the current working directory

string **getcwd** (void) \linebreak

Returns the current working directory.

opendir (PHP 3, PHP 4 >= 4.0.0)

open directory handle

resource **opendir** (string path) \linebreak

Returns a directory handle to be used in subsequent `closedir()`, `readdir()`, and `rewinddir()` calls.

If *path* is not a valid directory or the directory can not be opened due to permission restrictions or filesystem errors, **opendir()** returns `FALSE` and generates a PHP error. You can suppress the error output of **opendir()** by prepending '@' to the front of the function name.

Example 1. opendir() example

```
<?php

if ($dir = @opendir("/tmp")) {
    while (($file = readdir($dir)) !== false) {
        echo "$file\n";
    }
    closedir($dir);
}

?>
```

See also `is_dir()`.

readdir (PHP 3, PHP 4 >= 4.0.0)

read entry from directory handle

string **readdir** (resource dir_handle) \linebreak

Returns the filename of the next file from the directory. The filenames are returned in the order in which they are stored by the filesystem.

Please note the fashion in which **readdir()**'s return value is checked in the examples below. We are explicitly testing whether the return value is identical to (equal to and of the same type as--see Comparison Operators for more information) `FALSE` since otherwise, any directory entry whose name evaluates to `FALSE` will stop the loop.

Example 1. List all files in a directory

```
// Note that !== did not exist until 4.0.0-RC2
<?php
if ($handle = opendir('/path/to/files')) {
    echo "Directory handle: $handle\n";
    echo "Files:\n";
```

```

/* This is the correct way to loop over the directory. */
while (false !== ($file = readdir($handle))) {
    echo "$file\n";
}

/* This is the WRONG way to loop over the directory. */
while ($file = readdir($handle)) {
    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
if ($handle = opendir('.')) {
    while (false !== ($file = readdir($handle))) {
        if ($file != "." && $file != "..") {
            echo "$file\n";
        }
    }
    closedir($handle);
}
?>

```

See also `is_dir()`.

rewinddir (PHP 3, PHP 4 >= 4.0.0)

rewind directory handle

```
void rewinddir ( resource dir_handle ) \linebreak
```

Resets the directory stream indicated by *dir_handle* to the beginning of the directory.

XXV. DOM XML functions

Introduction

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

The domxml extension has been overhauled in PHP Version 4.3.0 in favour of a better compliance of the DOM standard. The extension still contains many old functions, but they should not be used anymore. Especially those non object oriented functions should be avoided.

The extension allows you to operate on an XML document with the DOM API. It also provides a function `domxml_xmltree()` to turn the complete XML document into a tree of PHP objects. Currently this tree should be considered read-only - you can modify it but this would not make any sense since `DomDocument_dump_mem()` cannot be applied to it. Therefore, if you want to read an XML file and write a modified version use the `DomDocument_create_element()`, `DomDocument_create_text_node()`, `set_attribute()`, etc. and finally `DomDocument_dump_mem()` functions.

Requirements

This extension is only available if PHP was configured with `--with-dom=[DIR]`, using the GNOME xml library (<http://www.xmlsoft.org>). You will need at least `libxml-2.2.7`

Deprecated functions

There are quite some functions which do not fit into the DOM standard and should not be used anymore as listed in the following table. The function `DomNode_append_child()` has changed its behaviour. It now actually adds a child and not a sibling. If this breaks your application use the non DOM function `DomNode_append_sibling()`.

Table 1. Deprecated functions and its replacements

Old function	New function
<code>xml doc</code>	<code>domxml_open_mem()</code>
<code>xml docfile</code>	<code>domxml_open_file()</code>
<code>domxml_new_xml doc</code>	<code>domxml_new_doc()</code>
<code>domxml_dump_mem</code>	<code>DomDocument_dump_mem()</code>

Old function	New function
domxml_dump_mem_file	DomDocument_dump_file()
DomDocument_dump_mem_file	DomDocument_dump_file()
DomDocument_add_root	DomDocument_create_element() followed by DomNode_append_child()
DomDocument_dtd	DomDocument_doctype()
DomDocument_root	DomDocument_document_element()
DomDocument_children	DomNode_child_nodes()
DomDocument_imported_node	No replacement.
DomNode_add_child	Create a new node with e.g. DomDocument_create_element() und add it with DomNode_append_child().
DomNode_children	DomNode_child_nodes()
DomNode_parent	DomNode_parent_node()
DomNode_new_child	Create a new node with e.g. DomDocument_create_element() und add it with DomNode_append_child().
DomNode_set_content	Create a new node with e.g. DomDocument_create_text_node() und add it with DomNode_append_child().
DomNode_get_content	Content is just a text node and can be accessed with DomNode_child_nodes().
DomNode_set_content	Content is just a text node and can be added with DomNode_append_child().

Constants

This module defines the following constants:

Table 2. 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

Constant	Value	Description
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	

Classes

The API of the module follows the DOM Level 2 standard as close as possible. Consequently the API is fully object oriented. It is a good idea to have the DOM standard available when using this module. Though the API is object oriented there are many functions which can be called in a non-object oriented way by passing the object to operate on as the first argument. These function are mainly to retain compatibility to older versions of the extension but are not encouraged to use anymore in new developments.

This API differs from the official DOM API in two points. First, all class attributes are implemented as functions with the same name and secondly the function names follow the PHP naming convention. This means that a DOM function `lastChild()` will be written as `last_child()`.

This module defines a number of classes, which are listed — including their method — in the following tables. Classes with an equivalent in the DOM Standard are named DOMxxx.

Table 3. List of classes

Class name	Parent classes
DomAttribute	DomNode
DomCDATA	DomNode
DomComment	DomCDATA : DomNode
DomDocument	DomNode
DomDocumentType	DomNode
DomElement	DomNode
DomEntity	DomNode
DomEntityReference	DomNode
DomProcessingInstruction	DomNode
DomText	DomCDATA : DomNode
Parser	Currently still called DomParser
XPathContext	

Table 4. DomDocument class (DomDocument : DomNode)

Method name	Function name	Remark
doctype	DomDocument_doctype()	
document_element	DomDocument_document_element()	
create_element	DomDocument_create_element()	
create_text_node	DomDocument_create_text_node()	
create_comment	DomDocument_create_comment()	
create_cdata_section	DomDocument_create_cdata_section()	
create_processing_instruction	DomDocument_create_processing_instruction()	
create_attribute	DomDocument_create_attribute()	
create_entity_reference	DomDocument_create_entity_reference()	
get_elements_by_tagname	DomDocument_get_elements_by_tagname()	
get_element_by_id	DomDocument_get_element_by_id()	
dump_mem	DomDocument_dump_mem()	not DOM standard
dump_file	DomDocument_dump_file()	not DOM standard
html_dump_mem	DomDocument_html_dump_mem()	not DOM standard
xpath_init	xpath_init	not DOM standard
xpath_new_context	xpath_new_context	not DOM standard
xptr_new_context	xptr_new_context	not DOM standard

Table 5. DomElement class (DomElement : DomNode)

Method name	Function name	Remark
tagname	DomElement_tagname()	
get_attribute	DomElement_get_attribute()	
set_attribute	DomElement_set_attribute()	
remove_attribute	DomElement_remove_attribute()	

Method name	Function name	Remark
get_attribute_node	DomElement_get_attribute_node()	
set_attribute_node	DomElement_set_attribute_node()	
get_elements_by_tagname	DomElement_get_elements_by_tagname()	
has_attribute	DomElement_has_attribute()	

Table 6. DomNode class

Method name	Remark
DomNode_node_name()	
DomNode_node_value()	
DomNode_node_type()	
DomNode_last_child()	
DomNode_first_child()	
DomNode_child_nodes()	
DomNode_previous_sibling()	
DomNode_next_sibling()	
DomNode_parent_node()	
DomNode_owner_document()	
DomNode_insert_before()	
DomNode_append_child()	
DomNode_append_sibling()	Not in DOM standard. This function emulates the former behaviour of DomNode_append_child().
DomNode_remove_child()	
DomNode_has_child_nodes()	
DomNode_has_attributes()	
DomNode_clone_node()	
DomNode_attributes()	
DomNode_unlink_node()	Not in DOM standard
DomNode_replace_node()	Not in DOM standard
DomNode_set_content()	Not in DOM standard, deprecated
DomNode_get_content()	Not in DOM standard, deprecated
DomNode_dump_node()	Not in DOM standard
DomNode_is_blank_node()	Not in DOM standard

Table 7. DomAttribute class (DomAttribute : DomNode)

Method name		Remark
name	DomAttribute_name()	
value	DomAttribute_value()	
specified	DomAttribute_specified()	

Table 8. DomProcessingInstruction class (DomProcessingInstruction : DomNode)

Method name	Function name	Remark
target	DomProcessingInstruction_target()	
data	DomProcessingInstruction_data()	

Table 9. Parser class

Method name	Function name	Remark
add_chunk	Parser_add_chunk()	
end	Parser_end()	

Table 10. XPathContext class

Method name	Function name	Remark
eval	XPathContext_eval()	
eval_expression	XPathContext_eval_expression()	
register_ns	XPathContext_register_ns()	

Table 11. DomDocumentType class (DomDocumentType : DomNode)

Method name	Function name	Remark
name	DomDocumentType_name()	
entities	DomDocumentType_entities()	
notations	DomDocumentType_notations()	

Method name	Function name	Remark
public_id	DomDocumentType_public_id()	
system_id	DomDocumentType_system_id()	
internal_subset	DomDocumentType_internal_subset()	

The classes DomDtd is derived from DomNode. DomComment is derived from DomCDATA

Examples

Many examples in this reference require a xml string. Instead of repeating this string in any example it will be put into a file and be included by each example. This include file is shown in the following example section. You could also create an xml document read it with **DomDocument_open_file()**.

Example 1. Include file example.inc with xml string

```
<?php
$xmlstr = "<?xml version='1.0' standalone='yes'?>
<!DOCTYPE chapter SYSTEM '/share/sgml/Norman_Walsh/db3xml10/db3xml10.dtd'
[ <!ENTITY sp \"spanish\">
]>
<!-- lsfj -->
<chapter language='en'><title language='en'>Title</title>
  <para language='ge'>
    &sp;
    <!-- comment -->
    <informaltable ID='findme' language='&sp;'>
      <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>
</chapter>";
?>
```

DomAttribute->name (unknown)

Returns name of attribute

bool **DomAttribute->name** (void) \linebreak

This function returns the name of the attribute.

See also DomAttribute_value().

DomAttribute->specified (unknown)

Checks if attribute is specified

bool **DomAttribute->specified** (void) \linebreak

Check DOM standard for a detailed explanation.

DomAttribute->value (unknown)

Returns value of attribute

bool **DomAttribute->value** (void) \linebreak

This function returns the value of the attribute.

See also DomAttribute_name().

DomDocument->add_root [deprecated] (unknown)

Adds a root node

resource **DomDocument->add_root** (string name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Adds a root element node to a dom document and returns the new node. The element name is given in the passed parameter.

Example 1. Creating a simple HTML document header

```

<?php
$doc = domxml_new_doc("1.0");
$root = $doc->add_root("HTML");
$head = $root->new_child("HEAD", "");
$head->new_child("TITLE", "Hier der Titel");
echo htmlentities($doc->dump_mem());
?>

```

DomDocument->create_attribute (unknown)

Create new attribute

object **DomDocument->create_attribute** (string name, string value) \linebreak

This function returns a new instance of class `DomAttribute`. The name of the attribute is the value of the first parameter. The value of the attribute is the value of the second parameter. This node will not show up in the document unless it is inserted with e.g. `DomNode_append_child()`.

The return value is false if an error occurred.

See also `DomNode_append_child()`, `DomDocument_create_element()`, **`DomDocument_create_text()`**, `DomDocument_create_cdata_section()`, `DomDocument_create_processing_instruction()`, `DomDocument_create_entity_reference()`, `DomNode_insert_before()`.

DomDocument->create_cdata_section (unknown)

Create new cdata node

string **DomDocument->create_cdata_section** (string content) \linebreak

This function returns a new instance of class `DomCData`. The content of the cdata is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. `DomNode_append_child()`.

The return value is false if an error occurred.

See also `DomNode_append_child()`, `DomDocument_create_element()`, **`DomDocument_create_text()`**, `DomDocument_create_attribute()`, `DomDocument_create_processing_instruction()`, `DomDocument_create_entity_reference()`, `DomNode_insert_before()`.

DomDocument->create_comment (unknown)

Create new comment node

object **DomDocument->create_comment** (string content) \linebreak

This function returns a new instance of class DomComment. The content of the comment is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. DomNode_append_child().

The return value is false if an error occurred.

See also DomNode_append_child(), DomDocument_create_element(), **DomDocument_create_text()**, DomDocument_create_attribute(), DomDocument_create_processing_instruction(), DomDocument_create_entity_reference(), DomNode_insert_before().

DomDocument->create_element (unknown)

Create new element node

object **DomDocument->create_element** (string name) \linebreak

This function returns a new instance of class DomElement. The tag name of the element is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. DomNode_append_child().

The return value is false if an error occurred.

See also DomNode_append_child(), **DomDocument_create_text()**, DomDocument_create_comment(), DomDocument_create_attribute(), DomDocument_create_processing_instruction(), DomDocument_create_entity_reference(), DomNode_insert_before().

DomDocument->create_entity_reference (unknown)

object **DomDocument->create_entity_reference** (string content) \linebreak

This function returns a new instance of class DomEntityReference. The content of the entity reference is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. DomNode_append_child().

The return value is false if an error occurred.

See also DomNode_append_child(), DomDocument_create_element(), **DomDocument_create_text()**, DomDocument_create_cdata_section(), DomDocument_create_processing_instruction(), DomDocument_create_attribute(), DomNode_insert_before().

DomDocument->create_processing_instruction (unknown)

Creates new PI node

string **DomDocument->create_processing_instruction** (string content) \linebreak

This function returns a new instance of class `DomCData`. The content of the pi is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. `DomNode_append_child()`.

The return value is false if an error occurred.

See also `DomNode_append_child()`, `DomDocument_create_element()`, **`DomDocument_create_text()`**, `DomDocument_create_cdata_section()`, `DomDocument_create_attribute()`, `DomDocument_create_entity_reference()`, `DomNode_insert_before()`.

DomDocument->create_text_node (unknown)

Create new text node

object **DomDocument->create_text_node** (string content) \linebreak

This function returns a new instance of class `DomText`. The content of the text is the value of the passed parameter. This node will not show up in the document unless it is inserted with e.g. `DomNode_append_child()`.

The return value is false if an error occurred.

See also `DomNode_append_child()`, `DomDocument_create_element()`, `DomDocument_create_comment()`, **`DomDocument_create_text()`**, `DomDocument_create_attribute()`, `DomDocument_create_processing_instruction()`, `DomDocument_create_entity_reference()`, `DomNode_insert_before()`.

DomDocument->doctype (unknown)

Returns the document type

object **DomDocument->doctype** (void) \linebreak

This function returns an object of class `DomDocumentType`. In versions of PHP before 4.3 this has been the class `Dtd`, but the DOM Standard does not know such a class.

See also the methods of class `DomDocumentType`.

DomDocument->document_element (unknown)

Returns root element node

object **DomDocument->document_element** (void) \linebreak

This function returns the root element node of a document.

The following example returns just the element with name CHAPTER and prints it. The other node -- the comment -- is not returned.

Example 1. Retrieving root element

```
<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$root = $dom->document_element();
print_r($root);
?>
```

DomDocument->dump_file (unknown)

Dumps the internal XML tree back into a file

string **DomDocument->dump_file** (string filename [, bool compressionmode [, bool format]]) \linebreak

Creates an XML document from the dom representation. This function usually is called after building a new dom document from scratch as in the example below. The *format* specifies whether the output should be neatly formatted, or not. The first parameter specifies the name of the filename and the second parameter, whether it should be compressed or not.

Example 1. Creating a simple HTML document header

```
<?php
$doc = domxml_new_doc("1.0");
$root = $doc->create_element("HTML");
$root = $doc->append_child($root);
$head = $doc->create_element("HEAD");
```

```

$head = $root->append_child($head);
$title = $doc->create_element("TITLE");
$title = $head->append_child($title);
$text = $doc->create_text_node("This is the title");
$text = $title->append_child($text);
$doc->dump_file("/tmp/test.xml", false, true);
?>

```

See also `DomDocument_dump_mem()` `DomDocument_html_dump_mem()`.

DomDocument->dump_mem (unknown)

Dumps the internal XML tree back into a string

string **DomDocument->dump_mem** ([bool format]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Creates an XML document from the dom representation. This function usually is called after building a new dom document from scratch as in the example below. The *format* specifies whether the output should be neatly formatted, or not.

Example 1. Creating a simple HTML document header

```

<?php
$doc = domxml_new_doc("1.0");
$root = $doc->create_element("HTML");
$root = $doc->append_child($root);
$head = $doc->create_element("HEAD");
$head = $root->append_child($head);
$title = $doc->create_element("TITLE");
$title = $head->append_child($title);
$text = $doc->create_text_node("This is the title");
$text = $title->append_child($text);
echo "<PRE>";
echo htmlentities($doc->dump_mem(true));
echo "</PRE>";
?>

```

Note: The first parameter was added in PHP 4.3.0.

See also `DomDocument_dump_file()`, `DomDocument_html_dump_mem()`.

DomDocument->get_element_by_id (unknown)

Searches for an element with a certain id

object **DomDocument->get_element_by_id** (string id) \linebreak

This function is similar to `DomDocument_get_elements_by_tagname()` but searches for an element with a given id. According to the DOM standard this requires a DTD which defines the attribute ID to be of type ID, though the current implementation simply does an xpath search for `"//*[@ID = '%s']"`. This does not comply to the DOM standard which requires to return null if it is not known which attribute is of type id. This behaviour is likely to be fixed, so do not rely on the current behaviour.

See also `DomDocument_get_elements_by_tagname()`

DomDocument->get_elements_by_tagname (unknown)

array **DomDocument->get_elements_by_tagname** (string name) \linebreak

See also `DomDocument_add_root()`

DomDocument->html_dump_mem (unknown)

Dumps the internal XML tree back into a string as HTML

string **DomDocument->html_dump_mem** (void) \linebreak

Creates an HTML document from the dom representation. This function usually is called after building a new dom document from scratch as in the example below.

Example 1. Creating a simple HTML document header

```
<?php
$doc = domxml_new_doc("1.0");
$root = $doc->create_element("HTML");
$root = $doc->append_child($root);
```



```

$head = $doc->create_element("HEAD");
$head = $root->append_child($head);
$title = $doc->create_element("TITLE");
$title = $head->append_child($title);
$text = $doc->create_text_node("This is the title");
$text = $title->append_child($text);
echo "<PRE>";
echo htmlentities($doc->html_dump_mem());
echo "</PRE>";
?>

```

See also `DomDocument_dump_file()`, `DomDocument_html_dump_mem()`.

DomDocumentType->entities (unknown)

Returns list of entities

array **DomDocumentType->entities** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomDocumentType->internal_subset (unknown)

Returns internal subset

bool **DomDocumentType->internal_subset** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomDocumentType->name (unknown)

Returns name of document type

string **DomDocumentType->name** (void) \linebreak

This function returns the name of the document type.

DomDocumentType->notations (unknown)

Returns list of notations

array **DomDocumentType->notations** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomDocumentType->public_id (unknown)

Returns public id of document type

string **DomDocumentType->public_id** (void) \linebreak

This function returns the public id of the document type.

The following example echos nothing.

Example 1. Retrieving the public id

```

<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$doctype = $dom->doctype();
echo $doctype->public_id();
?>

```

DomDocumentType->system_id (unknown)

Returns system id of document type

string **DomDocumentType->system_id** (void) \linebreak

Returns the system id of the document type.

The following example echos '/share/sgml/Norman_Walsh/db3xml10/db3xml10.dtd'.

Example 1. Retrieving the system id

```
<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$doctype = $dom->doctype();
echo $doctype->system_id();
?>
```

DomElement->get_attribute (unknown)

Returns value of attribute

object **DomElement->get_attribute** (string name) \linebreak

Returns the attribute with name *name* of the current node.

See also DomElement_set_attribute()

DomElement->get_attribute_node (unknown)

Returns value of attribute

object **DomElement->get_attribute_node** (object attr) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomElement->get_elements_by_tagname (unknown)

Adds new attribute

bool **DomElement->get_elements_by_tagname** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomElement->has_attribute (unknown)

Adds new attribute

bool **DomElement->has_attribute** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomElement->remove_attribute (unknown)

Adds new attribute

bool **DomElement->remove_attribute** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomElement->set_attribute (unknown)

Adds new attribute

bool **DomElement->set_attribute** (string name, string value) \linebreak

Sets an attribute with name *name* of the given value. If the attribute does not exist, it will be created.

Example 1. Setting an attribute

```

<?php
$doc = domxml_new_doc("1.0");
$node = $doc->create_element("para");
$newnode = $doc->append_child($node);
$newnode->set_attribute("align", "left");
?>

```

See also DomElement_get_attribute()

DomElement->set_attribute_node (unknown)

Adds new attribute

bool **DomElement->set_attribute_node** (object attr) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomElement->>tagname (unknown)

Returns name of element

string **DomElement->>tagname** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DOMNode->append_child (unknown)

Adds new child at the end of the children

object **DOMNode->append_child** (object newnode) \linebreak

This functions appends a child to an existing list of children or creates a new list of children. The child can be created with e.g. `DomDocument_create_element()`, `DomDocument_create_text()` etc. or simply by using any other node.

Before a new child is appended it is first duplicated. Therefore the new child is a completely new copy which can be modified without changing the node which was passed to this function. If the node passed has children itself, they will be duplicated as well, which makes it quite easy to duplicate large parts of a xml document. The return value is the appended child. If you plan to do further modifications on the appended child you must use the returned node.

The following example will add a new element node to a fresh document and sets the attribute "align" to "left".

Example 1. Adding a child

```
<?php
$doc = domxml_new_doc("1.0");
$node = $doc->create_element("para");
$newnode = $doc->append_child($node);
$newnode->set_attribute("align", "left");
?>
```

The above example could also be written as the following:

Example 2. Adding a child

```
<?php
$doc = domxml_new_doc("1.0");
$node = $doc->create_element("para");
$node->set_attribute("align", "left");
$newnode = $doc->append_child($node);
?>
```

A more complex example is the one below. It first searches for a certain element, duplicates it including its children and adds it as a sibling. Finally a new attribute is added to one of the children of the new sibling and the whole document is dumped.

Example 3. Adding a child

```
<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$elements = $dom->get_elements_by_tagname("informaltable");
print_r($elements);
$element = $elements[0];

$parent = $element->parent_node();
$newnode = $parent->append_child($element);
$children = $newnode->children();
$attr = $children[1]->set_attribute("align", "left");

echo "<PRE>";
$xmlfile = $dom->dump_mem();
echo htmlentities($xmlfile);
echo "</PRE>";
?>
```

The above example could also be done with `DomNode_insert_before()` instead of `DomNode_append_child()`.

See also `DomNode_insert_before()`.

DomNode->append_sibling (unknown)

Adds new sibling to a node

object **DomNode->append_sibling** (object newnode) \linebreak

This function appends a sibling to an existing node. The child can be created with e.g. `DomDocument_create_element()`, `DomDocument_create_text()` etc. or simply by using any other node.

Before a new sibling is added it is first duplicated. Therefore the new child is a completely new copy which can be modified without changing the node which was passed to this function. If the node passed

has children itself, they will be duplicated as well, which makes it quite easy to duplicate large parts of a xml document. The return value is the added sibling. If you plan to do further modifications on the added sibling you must use the returned node.

This function has been added to provide the behaviour of `DOMNode_append_child()` as it works till PHP 4.2.

See also `DOMNode_append_before()`.

DOMNode->attributes (unknown)

Returns list of attributes

array **DOMNode->attributes** (void) \linebreak

This function only returns an array of attributes if the node is of type `XML_ELEMENT_NODE`.

DOMNode->child-nodes (unknown)

Returns children of node

array **DOMNode->child_nodes** (void) \linebreak

Returns all children of the node.

See also `DOMNode_next_sibling()`, `DOMNode_previous_sibling()`.

DOMNode->clone_node (unknown)

Clones a node

object **DOMNode->clone_node** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DOMNode->dump_node (unknown)

Dumps a single node

string **DOMNode->dump_node** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

See also DomDocument_dump_mem().

DOMNode->first_child (unknown)

Returns first child of node

bool **DOMNode->first_child** (void) \linebreak

Returns the first child of the node.

See also DomNode_last_child(), DomNode_next_sibling(), DomNode_previous_sibling().

DOMNode->get_content (unknown)

Gets content of node

string **DOMNode->get_content** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DOMNode->has_attributes (unknown)

Checks if node has attributes

bool **DOMNode->has_attributes** (void) \linebreak

This function checks if the node has attributes.

See also DomNode_has_child_nodes().

DOMNode->has_child_nodes (unknown)

Checks if node has children

```
bool DOMNode->has_child_nodes ( void) \linebreak
```

This function checks if the node has children.

See also `DOMNode_child_nodes()`.

DOMNode->insert_before (unknown)

Inserts new node as child

```
object DOMNode->insert_before ( object newnode, object refnode) \linebreak
```

This function inserts the new node *newnode* right before the node *refnode*. The return value is the inserted node. If you plan to do further modifications on the appended child you must use the returned node.

`DOMNode_insert_before()` is very similar to `DOMNode_append_child()` as the following example shows which does the same as the example at `DOMNode_append_child()`.

Example 1. Adding a child

```
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$elements = $dom->get_elements_by_tagname("informaltable");
print_r($elements);
$element = $elements[0];

$newnode = $element->insert_before($element, $element);
$children = $newnode->children();
$attr = $children[1]->set_attribute("align", "left");

echo "<PRE>";
$xmlfile = $dom->dump_mem();
echo htmlentities($xmlfile);
echo "</PRE>";
```

See also `DomNode_append_child()`.

DomNode->is_blank_node (unknown)

Checks if node is blank

bool **DomNode->is_blank_node** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomNode->last_child (unknown)

Returns last child of node

object **DomNode->last_child** (void) \linebreak

Returns the last child of the node.

See also `DomNode_first_child()`, `DomNode_next_sibling()`, `DomNode_previous_sibling()`.

DomNode->next_sibling (unknown)

Returns the next sibling of node

object **DomNode->next_sibling** (void) \linebreak

This function returns the next sibling of the current node. If there is no next sibling it returns false. You can use this function to iterate over all children of a node as shown in the example.

Example 1. Iterate over children

```
<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}
```

```

$elements = $dom->get_elements_by_tagname("tbody");
$element = $elements[0];
$child = $element->first_child();

while($child) {
    print_r($child);
    $child = $child->next_sibling();
}
?>

```

See also `DomNode_previous_sibling()`.

DomNode->node_name (unknown)

Returns name of node

string **DomNode->node_name** (void) \linebreak

Returns name of the node. The name has different meanings for the different types of nodes as illustrated in the following table.

Table 1. Meaning of value

Type	Meaning
DomAttribute	value of attribute
DomAttribute	
DomCDATASection	#cdata-section
DomComment	#comment
DomDocument	#document
DomDocumentType	document type name
DomElement	tag name
DomEntity	name of entity
DomEntityReference	name of entity reference
DomNotation	notation name
DomProcessingInstruction	target
DomText	#text

DOMNode->node_type (unknown)

Returns type of node

int **DOMNode->node_type** (void) \linebreak

Returns the type of the node. All possible types are listed in the table in the introduction.

DOMNode->node_value (unknown)

Returns value of a node

string **DOMNode->node_value** (void) \linebreak

Returns value of the node. The value has different meanings for the different types of nodes as illustrated in the following table.

Table 1. Meaning of value

Type	Meaning
DomAttribute	value of attribute
DomAttribute	
DomCDATASection	content
DomComment	content of comment
DomDocument	null
DomDocumentType	null
DomElement	null
DomEntity	null
DomEntityReference	null
DomNotation	null
DomProcessingInstruction	entire content without target
DomText	content of text

DOMNode->owner_document (unknown)

Returns the document this node belongs to

object **DOMNode->owner_document** (void) \linebreak

This function returns the document the current node belongs to.

The following example will create two identical lists of children.

Example 1. Finding the document of a node

```

<?php
$doc = domxml_new_doc("1.0");
$node = $doc->create_element("para");
$node = $doc->append_child($node);
$children = $doc->children();
print_r($children);

$doc2 = $node->owner_document();
$children = $doc2->children();
print_r($children);
?>

```

See also `DOMNode_insert_before()`.

DOMNode->parent_node (unknown)

Returns the parent of the node

object **DOMNode->parent_node** (void) \linebreak

This function returns the parent node.

The following example will show two identical lists of children.

Example 1. Finding the document of a node

```

<?php
$doc = domxml_new_doc("1.0");
$node = $doc->create_element("para");
$node = $doc->append_child($node);
$children = $doc->children();
print_r($children);

$doc2 = $node->parent_node();
$children = $doc2->children();
print_r($children);
?>

```

DOMNode->prefix (unknown)

Returns name space prefix of node

string **DOMNode->prefix** (void) \linebreak

Returns the name space prefix of the node.

DOMNode->previous_sibling (unknown)

Returns the previous sibling of node

object **DOMNode->previous_sibling** (void) \linebreak

This function returns the previous sibling of the current node.

See also DomNode_next_sibling().

DOMNode->remove_child (unknown)

Removes child from list of children

object **DOMNode->remove_child** (object oldchild) \linebreak

This functions removes a child from a list of children. If child cannot be removed or is not a child the function will return false. If the child could be removed the functions returns the old child.

Example 1. Removing a child

```

<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$elements = $dom->get_elements_by_tagname("tbody");
$element = $elements[0];
$children = $element->child_nodes();
$child = $element->remove_child($children[0]);

echo "<PRE>";
$xmlfile = $dom->dump_mem(true);
echo htmlentities($xmlfile);
echo "</PRE>";

```

?>

See also `DomNode_append_child()`.

DomNode->replace_child (unknown)

Replaces a child

object **DomNode->replace_child** (object *oldnode*, object *newnode*) \linebreak

This function replaces the child *oldnode* with the passed new node. If the new node is already a child it will not be added a second time. If the old node cannot be found the function returns false. If the replacement succeeds the old node is returned.

See also `DomNode_append_child()`

DomNode->replace_node (unknown)

Replaces node

object **DomNode->replace_node** (object *newnode*) \linebreak

This function replaces an existing node with the passed new node. Before the replacement *newnode* is copied if it has a parent to make sure a node which is already in the document will not be inserted a second time. This behaviour enforces doing all modifications on the node before the replacement or to refetch the inserted node afterwards with functions like `DomNode_first_child()`, `DomNode_child_nodes()` etc..

See also `DomNode_append_child()`

DomNode->set_content (unknown)

Sets content of node

bool **DomNode->set_content** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DOMNode->set_name (unknown)

Sets name of node

bool **DOMNode->set_name** (void) \linebreak

Sets name of node.

See also DomNode_node_name().

DOMNode->unlink_node (unknown)

Deletes node

object **DOMNode->unlink_node** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomProcessingInstruction->data (unknown)

Returns data of pi node

string **DomProcessingInstruction->data** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

DomProcessingInstruction->target (unknown)

Returns target of pi node

string **DomProcessingInstruction->target** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

domxml_new_doc (unknown)

Creates new empty XML document

object **domxml_new_doc** (string version) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Creates a new dom document from scratch and returns it.

See also DomDocument_add_root()

domxml_open_file (unknown)

Creates a DOM object from XML file

object **domxml_open_file** (string filename) \linebreak

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.

Example 1. Opening a xml document from a file

```
<?php

if(!$dom = domxml_open_file("example.xml")) {
    echo "Error while parsing the document\n";
    exit;
}

$root = $dom->document_element();
?>
```

See also `domxml_open_mem()`, `domxml_new_doc()`.

domxml_open_mem (unknown)

Creates a DOM object of an XML document

object **domxml_open_mem** (string *str*) \linebreak

The function parses the XML document in *str* and returns an object of class "Dom document", having the properties as listed above. This function, `domxml_open_file()` or `domxml_new_doc()` must be called before any other function calls.

Example 1. Opening a xml document in a string

```
<?php
include("example.inc");

if(!$dom = domxml_open_mem($xmlstr)) {
    echo "Error while parsing the document\n";
    exit;
}

$root = $dom->document_element();
?>
```

See also `domxml_open_file()`, `domxml_new_doc()`.

domxml_version (PHP 4 >= 4.1.0)

Get XML library version

string **domxml_version** (void) \linebreak

This function returns the version of the XML library version currently used.

domxml_xmltree (unknown)

Creates a tree of PHP objects from an XML document

object **domxml_xmltree** (string *str*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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. However this function may be valuable if you want to read a file and investigate the content.

xpath_eval (PHP 4 >= 4.0.4)

Evaluates the XPath Location Path in the given string

array **xpath_eval** (object *xpath context*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

See also `xpath_new_context()`

xpath_eval_expression (PHP 4 >= 4.0.4)

Evaluates the XPath Location Path in the given string

array **xpath_eval_expression** (object *xpath_context*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

See also `xpath_eval()`

xpath_new_context (PHP 4 >= 4.0.4)

Creates new xpath context

object **xpath_new_context** (object dom document) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

See also `xpath_eval()`

xptr_eval (PHP 4 >= 4.0.4)

Evaluate the XPtr Location Path in the given string

int **xptr_eval** ([object xpath_context, string eval_str]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

xptr_new_context (PHP 4 >= 4.0.4)

Create new XPath Context

string **xptr_new_context** ([object doc_handle]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

XXVI. .NET functions

dotnet_load (unknown)

Loads a DOTNET module

```
int dotnet_load ( string assembly_name [, string datatype_name [, int codepage]]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

XXVII. 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 >= 4.0.0)

send an error message somewhere

```
int error_log ( string message [, int message_type [, string destination [, string extra_headers]]) \linebreak
```

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> .

Warning

Remote debugging via TCP/IP is a PHP 3 feature that is *not* available in PHP 4.

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 >= 4.0.0)

set which PHP errors are reported

```
int error_reporting ( [int level] ) \linebreak
```

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 of the constants to get their meanings:

Table 1. error_reporting() bit values

value	constant
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING

value	constant
64	E_COMPILE_ERROR
128	E_COMPILE_WARNING
256	E_USER_ERROR
512	E_USER_WARNING
1024	E_USER_NOTICE
2047	E_ALL

Example 2. error_reporting() examples

```
// Turn off all error reporting
error_reporting(0);

// Report simple running errors
error_reporting (E_ERROR | E_WARNING | E_PARSE);

// Reporting E_NOTICE can be good too (to report uninitialized
// variables or catch variable name misspellings)
error_reporting (E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Report all PHP errors (use bitwise 63 in PHP 3)
error_reporting (E_ALL);
```

restore_error_handler (PHP 4)

Restores the previous error handler function

```
void restore_error_handler ( void) \linebreak
```

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)

Sets a user-defined error handler function.

string **set_error_handler** (string error_handler) \linebreak

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] $errstr<br>\n";
            break;
        default:
            echo "Unkown error type: [$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)

Generates a user-level error/warning/notice message

```
void trigger_error ( string error_msg [, int error_type]) \linebreak
```

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 >= 4.0.0)

Generates a user-level error/warning/notice message

void **user_error** (string error_msg [, int error_type]) \linebreak

This is an alias for the function `trigger_error()`.

See also `error_reporting()`, `set_error_handler()`, `restore_error_handler()`, and `trigger_error()`

XXVIII. FrontBase Functions

These functions allow you to access FrontBase database servers. In order to have these functions available, you must compile php with fbsql support by using the `--with-fbsql` option. If you use this option without specifying the path to fbsql, php will search for the fbsql client libraries in the default installation location for the platform. Users who installed FrontBase in a non standard directory should always specify the path to fbsql: `--with-fbsql=/path/to/fbsql`. This will force php to use the client libraries installed by FrontBase, avoiding any conflicts.

More information about FrontBase can be found at <http://www.frontbase.com/>.

Documentation for FrontBase can be found at <http://www.frontbase.com/cgi-bin/WebObjects/FrontBase.woa/wa/productsPage?currentPage=Documentation>.

Frontbase support has been added to PHP 4.0.6.

fbsql_affected_rows (PHP 4 >= 4.0.6)

Get number of affected rows in previous FrontBase operation

int **fbsql_affected_rows** ([int link_identifier]) \linebreak

fbsql_affected_rows() returns the number of rows affected by the last INSERT, UPDATE or DELETE query associated with *link_identifier*. If the link identifier isn't specified, the last link opened by **fbsql_connect()** is assumed.

Note: If you are using transactions, you need to call **fbsql_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.

Note: When using UPDATE, FrontBase will not update columns where the new value is the same as the old value. This creates the possibility that **fbsql_affected_rows()** may not actually equal the number of rows matched, only the number of rows that were literally affected by the query.

If the last query failed, this function will return -1.

See also: **fbsql_num_rows()**.

fbsql_autocommit (PHP 4 >= 4.0.6)

Enable or disable autocommit

bool **fbsql_autocommit** (resource link_identifier [, bool OnOff]) \linebreak

fbsql_autocommit() returns the current autocommit status. if the optional OnOff parameter is given the auto commit status will be changed. With OnOff set to TRUE each statement will be committed automatically, if no errors was found. With OnOff set to FALSE the user must commit or rollback the transaction using either **fbsql_commit()** or **fbsql_rollback()**.

See also: **fbsql_commit()** and **fbsql_rollback()**

fbsql_change_user (unknown)

Change logged in user of the active connection

resource **fbsql_change_user** (string user, string password [, string database [, int link_identifier]]) \linebreak

fbsql_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 to current database after the user has been changed. If the new user and password authorization fails, the current connected user stays active.

fbsql_close (PHP 4 >= 4.0.6)

Close FrontBase connection

boolean **fbsql_close** ([resource `link_identifier`]) \linebreak

Returns: TRUE on success, FALSE on error.

fbsql_close() closes the connection to the FrontBase server that's associated with the specified link identifier. If `link_identifier` isn't specified, the last opened link is used.

Using **fbsql_close()** isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

Example 1. fbsql_close() example

```
<?php
    $link = fbsql_connect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
    print ("Connected successfully");
    fbsql_close ($link);
?>
```

See also: `fbsql_connect()` and `fbsql_pconnect()`.

fbsql_commit (PHP 4 >= 4.0.6)

Commits a transaction to the database

bool **fbsql_commit** ([resource `link_identifier`]) \linebreak

Returns: TRUE on success, FALSE on failure.

fbsql_commit() ends the current transaction by writing all inserts, updates and deletes to the disk and unlocking all row and table locks held by the transaction. This command is only needed if `autocommit` is set to false.

See also: `fbsql_autocommit()` and `fbsql_rollback()`

fbsql_connect (PHP 4 >= 4.0.6)

Open a connection to a FrontBase Server

resource **fbsql_connect** ([string hostname [, string username [, string password]]]) \linebreak

Returns a positive FrontBase link identifier on success, or an error message on failure.

fbsql_connect() establishes a connection to a FrontBase server. The following defaults are assumed for missing optional parameters: *hostname* = 'NULL', *username* = '_SYSTEM' and *password* = empty password.

If a second call is made to **fbsql_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 **fbsql_close()**.

Example 1. fbsql_connect() example

```
<?php
    $link = fbsql_connect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
    print ("Connected successfully");
    fbsql_close ($link);
?>
```

See also **fbsql_pconnect()** and **fbsql_close()**.

fbsql_create_blob (PHP 4 >= 4.2.0)

Create a BLOB

string **fbsql_create_blob** (string blob_data [, resource link_identifier]) \linebreak

Returns: A resource handle to the newly created blob.

fbsql_create_blob() creates a blob from *blob_data*. The returned resource handle can be used with insert and update commands to store the blob in the database.

Example 1. fbsql_create_blob() example

```
<?php
    $link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
```

```

$filename = "blobfile.bin";
$fp = fopen($filename, "rb");
$blobdata = fread($fp, filesize($filename));
fclose($fp);

$blobHandle = fbsql_create_blob($blobdata, $link);

$sql = "INSERT INTO BLOB_TABLE (BLOB_COLUMN) VALUES ($blobHandle)";
$rs = fbsql_query($sql, $link);
?>

```

See also: `fbsql_create_clob()`, `fbsql_read_blob()`, `fbsql_read_clob()`, and `fbsql_set_lob_mode()`.

fbsql_create_clob (PHP 4 >= 4.2.0)

Create a CLOB

string **fbsql_create_clob** (string clob_data [, resource link_identifier]) \linebreak

Returns: A resource handle to the newly created CLOB.

fbsql_create_clob() creates a clob from clob_data. The returned resource handle can be used with insert and update commands to store the clob in the database.

Example 1. fbsql_create_clob() example

```

<?php
$link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
    or die ("Could not connect");
$filename = "clob_file.txt";
$fp = fopen($filename, "rb");
$clobdata = fread($fp, filesize($filename));
fclose($fp);

$clobHandle = fbsql_create_clob($clobdata, $link);

$sql = "INSERT INTO CLOB_TABLE (CLOB_COLUMN) VALUES ($clobHandle)";
$rs = fbsql_query($sql, $link);
?>

```

See also: `fbsql_create_blob()`, `fbsql_read_blob()`, `fbsql_read_clob()`, and `fbsql_set_lob_mode()`.

fbsql_create_db (PHP 4 >= 4.0.6)

Create a FrontBase database

bool **fbsql_create_db** (string database name [, resource link_identifier]) \linebreak

fbsql_create_db() attempts to create a new database on the server associated with the specified link identifier.

Example 1. fbsql_create_db() example

```
<?php
$link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
    or die ("Could not connect");
if (fbsql_create_db ("my_db")) {
    print("Database created successfully\n");
} else {
    printf("Error creating database: %s\n", fbsql_error ());
}
?>
```

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

fbsql_data_seek (PHP 4 >= 4.0.6)

Move internal result pointer

bool **fbsql_data_seek** (resource result_identifier, int row_number) \linebreak

Returns: TRUE on success, FALSE on failure.

fbsql_data_seek() moves the internal row pointer of the FrontBase result associated with the specified result identifier to point to the specified row number. The next call to [fbsql_fetch_row\(\)](#) would return that row.

Row_number starts at 0.

Example 1. fbsql_data_seek() example

```
<?php
$link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
    or die ("Could not connect");

fbsql_select_db ("samp_db")
    or die ("Could not select database");
```

```

$query = "SELECT last_name, first_name FROM friends;";
$result = fbsql_query ($query)
    or die ("Query failed");

# fetch rows in reverse order

for ($i = fbsql_num_rows ($result) - 1; $i >=0; $i--) {
    if (!fbsql_data_seek ($result, $i)) {
        printf ("Cannot seek to row %d\n", $i);
        continue;
    }

    if(!($row = fbsql_fetch_object ($result)))
        continue;

    printf("%s %s<BR>\n", $row->last_name, $row->first_name);
}

fbsql_free_result ($result);
?>

```

fbsql_database (PHP 4 >= 4.0.6)

Get or set the database name used with a connection

string **fbsql_database** (resource link_identifier [, string database]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

fbsql_database_password (PHP 4 >= 4.0.6)

Sets or retrieves the password for a FrontBase database

string **fbsql_database_password** (resource link_identifier [, string database_password]) \linebreak

Returns: The database password associated with the link identifier.

fbsql_database_password() sets and retrieves the database password used by the connection. if a database is protected by a database password, the user must call this function before calling `fbsql_select_db()`. if the second optional parameter is given the function sets the database password for

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 `fbsql_connect()` was called, and use it.

This function does not change the database password in the database nor can it be used to retrieve the database password for a database.

Example 1. `fbsql_create_clob()` example

```
<?php
    $link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
    fbsql_database_password($link, "secret db password");
    fbsql_select_db($database, $link);
?>
```

See also: `fbsql_connect()`, `fbsql_pconnect()` and `fbsql_select_db()`.

`fbsql_db_query` (PHP 4 >= 4.0.6)

Send a FrontBase query

resource `fbsql_db_query` (string database, string query [, resource link_identifier]) \linebreak

Returns: A positive FrontBase result identifier to the query result, or `FALSE` on error.

`fbsql_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 FrontBase server and if no such link is found it'll try to create one as if `fbsql_connect()` was called with no arguments

See also `fbsql_connect()`.

`fbsql_db_status` (PHP 4 >= 4.1.0)

Get the status for a given database

int `fbsql_db_status` (string database_name [, resource link_identifier]) \linebreak

Returns: An integer value with the current status.

`fbsql_db_status()` requests the current status of the database specified by `database_name`. If the `link_identifier` is omitted the default `link_identifier` will be used.

The return value can be one of the following constants:

- `FALSE` - The exec handler for the host was invalid. This error will occur when the `link_identifier` connects directly to a database by using a port number. `FBExec` can be available on the server but no connection has been made for it.
- `FBSQL_UNKNOWN` - The Status is unknown.
- `FBSQL_STOPPED` - The database is not running. Use `fbsql_start_db()` to start the database.
- `FBSQL_STARTING` - The database is starting.
- `FBSQL_RUNNING` - The database is running and can be used to perform SQL operations.
- `FBSQL_STOPPING` - The database is stopping.
- `FBSQL_NOEXEC` - `FBExec` is not running on the server and it is not possible to get the status of the database.

See also: `fbsql_start_db()` and `fbsql_stop_db()`.

fbsql_drop_db (PHP 4 >= 4.0.6)

Drop (delete) a FrontBase database

```
bool fbsql_drop_db ( string database_name [, resource link_identifier] ) \linebreak
```

Returns: `TRUE` on success, `FALSE` on failure.

fbsql_drop_db() attempts to drop (remove) an entire database from the server associated with the specified link identifier.

fbsql_errno (PHP 4 >= 4.0.6)

Returns the numerical value of the error message from previous FrontBase operation

```
int fbsql_errno ( [resource link_identifier] ) \linebreak
```

Returns the error number from the last `fbsql` function, or 0 (zero) if no error occurred.

Errors coming back from the `fbsql` database backend don't issue warnings. Instead, use **fbsql_errno()** to retrieve the error code. Note that this function only returns the error code from the most recently executed `fbsql` function (not including `fbsql_error()` and **fbsql_errno()**), so if you want to use it, make sure you check the value before calling another `fbsql` function.

```
<?php
fbsql_connect("marliesle");
echo fbsql_errno().": ".fbsql_error()."<BR>";
fbsql_select_db("nonexistentdb");
echo fbsql_errno().": ".fbsql_error()."<BR>";
```



```
$conn = fbsql_query("SELECT * FROM nonexistenttable;");
echo fbsql_errno().": ".fbsql_error()."<BR>";
?>
```

See also: `fbsql_error()` and `fbsql_warnings()`.

fbsql_error (PHP 4 >= 4.0.6)

Returns the text of the error message from previous FrontBase operation

string **fbsql_error** ([resource link_identifier]) \linebreak

Returns the error text from the last `fbsql` function, or "" (the empty string) if no error occurred.

Errors coming back from the `fbsql` database backend don't issue warnings. Instead, use **fbsql_error()** to retrieve the error text. Note that this function only returns the error text from the most recently executed `fbsql` function (not including **fbsql_error()** and `fbsql_errno()`), so if you want to use it, make sure you check the value before calling another `fbsql` function.

```
<?php
fbsql_connect("marliesle");
echo fbsql_errno().": ".fbsql_error()."<BR>";
fbsql_select_db("nonexistentdb");
echo fbsql_errno().": ".fbsql_error()."<BR>";
$conn = fbsql_query("SELECT * FROM nonexistenttable;");
echo fbsql_errno().": ".fbsql_error()."<BR>";
?>
```

See also: `fbsql_errno()` and `fbsql_warnings()`.

fbsql_fetch_array (PHP 4 >= 4.0.6)

Fetch a result row as an associative array, a numeric array, or both

array **fbsql_fetch_array** (resource result [, int result_type]) \linebreak

Returns an array that corresponds to the fetched row, or `FALSE` if there are no more rows.

fbsql_fetch_array() is an extended version of **fbsql_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 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 **fbsql_fetch_array()** is NOT significantly slower than using **fbsql_fetch_row()**, while it provides a significant added value.

The optional second argument *result_type* in **fbsql_fetch_array()** is a constant and can take the following values: **FBSQL_ASSOC**, **FBSQL_NUM**, and **FBSQL_BOTH**.

For further details, see also **fbsql_fetch_row()** and **fbsql_fetch_assoc()**.

Example 1. **fbsql_fetch_array()** example

```
<?php
fbsql_connect ($host, $user, $password);
$result = fbsql_db_query ("database","select user_id, fullname from table");
while ($row = fbsql_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: ".$row[1]."<br>\n";
}
fbsql_free_result ($result);
?>
```

fbsql_fetch_assoc (PHP 4 >= 4.0.6)

Fetch a result row as an associative array

array **fbsql_fetch_assoc** (resource result) \linebreak

Returns an associative array that corresponds to the fetched row, or **FALSE** if there are no more rows.

fbsql_fetch_assoc() is equivalent to calling **fbsql_fetch_array()** with **FBSQL_ASSOC** for the optional second parameter. It only returns an associative array. This is the way **fbsql_fetch_array()** originally worked. If you need the numeric indices as well as the associative, use **fbsql_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 `fbsql_fetch_array()` and have it return the numeric indices as well.

An important thing to note is that using `fbsql_fetch_assoc()` is NOT significantly slower than using `fbsql_fetch_row()`, while it provides a significant added value.

For further details, see also `fbsql_fetch_row()` and `fbsql_fetch_array()`.

Example 1. `fbsql_fetch_assoc()` example

```
<?php
fbsql_connect ($host, $user, $password);
$result = fbsql_db_query ("database","select * from table");
while ($row = fbsql_fetch_assoc ($result)) {
    echo $row["user_id"];
    echo $row["fullname"];
}
fbsql_free_result ($result);
?>
```

`fbsql_fetch_field` (PHP 4 >= 4.0.6)

Get column information from a result and return as an object

object `fbsql_fetch_field` (resource result [, int field_offset]) \linebreak

Returns an object containing field information.

`fbsql_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 `fbsql_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
- type - the type of the column

Example 1. fbsql_fetch_field() example

```

<?php
fbsql_connect ($host, $user, $password)
    or die ("Could not connect");
$result = fbsql_db_query ("database", "select * from table")
    or die ("Query failed");
# get column metadata
$i = 0;
while ($i < fbsql_num_fields ($result)) {
    echo "Information for column $i:<BR>\n";
    $meta = fbsql_fetch_field ($result);
    if (!$meta) {
        echo "No information available<BR>\n";
    }
    echo "<PRE>
max_length:    $meta->max_length
name:         $meta->name
not_null:     $meta->not_null
table:       $meta->table
type:        $meta->type
</PRE>";
    $i++;
}
fbsql_free_result ($result);
?>

```

See also `fbsql_field_seek()`.

fbsql_fetch_lengths (PHP 4 >= 4.0.6)

Get the length of each output in a result

array **fbsql_fetch_lengths** ([resource result]) \linebreak

Returns: An array that corresponds to the lengths of each field in the last row fetched by `fbsql_fetch_row()`, or `FALSE` on error.

fbsql_fetch_lengths() stores the lengths of each result column in the last row returned by `fbsql_fetch_row()`, `fbsql_fetch_array()` and `fbsql_fetch_object()` in an array, starting at offset 0.

See also: `fbsql_fetch_row()`.

fbsql_fetch_object (PHP 4 >= 4.0.6)

Fetch a result row as an object

object **fbsql_fetch_object** (resource result [, int result_type]) \linebreak

Returns an object with properties that correspond to the fetched row, or `FALSE` if there are no more rows.

fbsql_fetch_object() is similar to `fbsql_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: `FBSQL_ASSOC`, `FBSQL_NUM`, and `FBSQL_BOTH`.

Speed-wise, the function is identical to `fbsql_fetch_array()`, and almost as quick as `fbsql_fetch_row()` (the difference is insignificant).

Example 1. fbsql_fetch_object() example

```
<?php
fbsql_connect ($host, $user, $password);
$result = fbsql_db_query ("database", "select * from table");
while ($row = fbsql_fetch_object ($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
fbsql_free_result ($result);
?>
```

See also: `fbsql_fetch_array()` and `fbsql_fetch_row()`.

fbsql_fetch_row (PHP 4 >= 4.0.6)

Get a result row as an enumerated array

array **fbsql_fetch_row** (resource result) \linebreak

Returns: An array that corresponds to the fetched row, or `FALSE` if there are no more rows.

fbsql_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 **fbsql_fetch_row()** would return the next row in the result set, or `FALSE` if there are no more rows.

See also: `fbsql_fetch_array()`, `fbsql_fetch_object()`, `fbsql_data_seek()`, `fbsql_fetch_lengths()`, and `fbsql_result()`.

fbsql_field_flags (PHP 4 >= 4.0.6)

Get the flags associated with the specified field in a result

string **fbsql_field_flags** (resource result, int field_offset) \linebreak

fbsql_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()`.

fbsql_field_len (PHP 4 >= 4.0.6)

Returns the length of the specified field

int **fbsql_field_len** (resource result, int field_offset) \linebreak

fbsql_field_len() returns the length of the specified field.

fbsql_field_name (PHP 4 >= 4.0.6)

Get the name of the specified field in a result

string **fbsql_field_name** (resource result, int field_index) \linebreak

fbsql_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. fbsql_field_name() example

```
// The users table consists of three fields:
//  user_id
//  username
//  password.

$res = fbsql_db_query("users", "select * from users", $link);
```

```
echo fbsql_field_name($res, 0) . "\n";
echo fbsql_field_name($res, 2);
```

The above example would produce the following output:

```
user_id
password
```

fbsql_field_seek (PHP 4 >= 4.0.6)

Set result pointer to a specified field offset

```
bool fbsql_field_seek ( resource result, int field_offset) \linebreak
```

Seeks to the specified field offset. If the next call to `fbsql_fetch_field()` doesn't include a field offset, the field offset specified in `fbsql_field_seek()` will be returned.

See also: `fbsql_fetch_field()`.

fbsql_field_table (PHP 4 >= 4.0.6)

Get name of the table the specified field is in

```
string fbsql_field_table ( resource result, int field_offset) \linebreak
```

Returns the name of the table that the specified field is in.

fbsql_field_type (PHP 4 >= 4.0.6)

Get the type of the specified field in a result

```
string fbsql_field_type ( resource result, int field_offset) \linebreak
```

`fbsql_field_type()` is similar to the `fbsql_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 FrontBase documentation (<http://www.frontbase.com/cgi-bin/WebObjects/FrontBase.woa/wa/productsPage?currentPage=Documentation>).

Example 1. `fbsql_field_type()` example

```
<?php

fbsql_connect ("localhost", "_SYSTEM", "");
fbsql_select_db ("wisconsin");
$result = fbsql_query ("SELECT * FROM onek;");
$fields = fbsql_num_fields ($result);
$rows   = fbsql_num_rows ($result);
$i = 0;
$table = fbsql_field_table ($result, $i);
echo "Your '". $table. "' table has ". $fields. " fields and ". $rows. " records <BR>";
echo "The table has the following fields <BR>";
while ($i < $fields) {
    $type = fbsql_field_type ($result, $i);
    $name = fbsql_field_name ($result, $i);
    $len  = fbsql_field_len ($result, $i);
    $flags = fbsql_field_flags ($result, $i);
    echo $type. " ". $name. " ". $len. " ". $flags. "<BR>";
    $i++;
}
fbsql_close();

?>
```

fbsql_free_result (PHP 4 >= 4.0.6)

Free result memory

bool **fbsql_free_result** (int result) \linebreak

fbsql_free_result() will free all memory associated with the result identifier *result*.

fbsql_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.

fbsql_get_autostart_info (PHP 4 >= 4.1.0)

No description given yet

array **fbsql_get_autostart_info** ([resource link_identifier]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

fbsql_hostname (PHP 4 >= 4.0.6)

Get or set the host name used with a connection

string **fbsql_hostname** (resource link_identifier [, string host_name]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

fbsql_insert_id (PHP 4 >= 4.0.6)

Get the id generated from the previous INSERT operation

int **fbsql_insert_id** ([resource link_identifier]) \linebreak

fbsql_insert_id() returns the ID generated for a column defined as DEFAULT UNIQUE by the previous INSERT query using the given *link_identifier*. If *link_identifier* isn't specified, the last opened link is assumed.

fbsql_insert_id() returns 0 if the previous query does not generate an DEFAULT UNIQUE value. If you need to save the value for later, be sure to call **fbsql_insert_id()** immediately after the query that generates the value.

Note: The value of the FrontBase SQL function `LAST_INSERT_ID()` always contains the most recently generated DEFAULT UNIQUE value, and is not reset between queries.

fbsql_list_dbs (PHP 4 >= 4.0.6)

List databases available on a FrontBase server

resource **fbsql_list_dbs** ([resource link_identifier]) \linebreak

fbsql_list_dbs() will return a result pointer containing the databases available from the current fbsql daemon. Use the **fbsql_tablename()** function to traverse this result pointer.

Example 1. fbsql_list_dbs() example

```
$link = fbsql_connect('localhost', 'myname', 'secret');
$db_list = fbsql_list_dbs($link);

while ($row = fbsql_fetch_object($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 **fbsql_fetch_row()** or other similar functions.

fbsql_list_fields (PHP 4 >= 4.0.6)

List FrontBase result fields

resource **fbsql_list_fields** (string database_name, string table_name [, resource link_identifier]) \linebreak

fbsql_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 **fbsql_field_flags()**, **fbsql_field_len()**, **fbsql_field_name()**, and **fbsql_field_type()**.

A result identifier is a positive integer. The function returns -1 if a error occurs. A string describing the error will be placed in `$phperrormsg`, and unless the function was called as `@fbsql()` then this error string will also be printed out.

Example 1. `fbsql_list_fields()` example

```
$link = fbsql_connect('localhost', 'myname', 'secret');

$fields = fbsql_list_fields("database1", "table1", $link);
$numcols = fbsql_num_fields($fields);

for ($i = 0; $i < $numcols; $i++) {
    echo fbsql_field_name($fields, $i) . "\n";
}
```

The above example would produce the following output:

```
field1
field2
field3
...
```

`fbsql_list_tables` (PHP 4 >= 4.0.6)

List tables in a FrontBase database

resource **fbsql_list_tables** (string database [, resource link_identifier]) \linebreak

fbsql_list_tables() takes a database name and returns a result pointer much like the `fbsql_db_query()` function. The `fbsql_tablename()` function should be used to extract the actual table names from the result pointer.

`fbsql_next_result` (PHP 4 >= 4.0.6)

Move the internal result pointer to the next result

bool **fbsql_next_result** (int result_id) \linebreak

When sending more than one SQL statement to the server or executing a stored procedure with multiple results will cause the server to return multiple result sets. This function will test for additional results available from the server. If an additional result set exists it will free the existing result set and prepare to fetch the words from the new result set. The function will return TRUE if an additional result set was available or FALSE otherwise.

Example 1. fbsql_next_result() example

```
<?php
$link = fbsql_connect ("localhost", "_SYSTEM", "secret");
fbsql_select_db("MyDB", $link);
$SQL = "Select * from table1; select * from table2;";
$rs = fbsql_query($SQL, $link);
do {
    while ($row = fbsql_fetch_row($rs)) {
    }
} while (fbsql_next_result($rs));
fbsql_free_result($rs);
fbsql_close ($link);
?>
```

fbsql_num_fields (PHP 4 >= 4.0.6)

Get number of fields in result

int **fbsql_num_fields** (resource result) \linebreak

fbsql_num_fields() returns the number of fields in a result set.

See also: fbsql_db_query(), fbsql_query(), fbsql_fetch_field(), and fbsql_num_rows().

fbsql_num_rows (PHP 4 >= 4.0.6)

Get number of rows in result

int **fbsql_num_rows** (resource result) \linebreak

fbsql_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 query, use fbsql_affected_rows().

Example 1. fbsql_num_rows() example

```

<?php

$link = fbsql_connect("localhost", "username", "password");
fbsql_select_db("database", $link);

$result = fbsql_query("SELECT * FROM table1;", $link);
$num_rows = fbsql_num_rows($result);

echo "$num_rows Rows\n";

?>

```

See also: `fbsql_affected_rows()`, `fbsql_connect()`, `fbsql_select_db()`, and `fbsql_query()`.

fbsql_password (PHP 4 >= 4.0.6)

Get or set the user password used with a connection

string **fbsql_password** (resource link_identifier [, string password]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

fbsql_pconnect (PHP 4 >= 4.0.6)

Open a persistent connection to a FrontBase Server

resource **fbsql_pconnect** ([string hostname [, string username [, string password]]]) \linebreak

Returns: A positive FrontBase persistent link identifier on success, or `FALSE` on error.

fbsql_pconnect() establishes a connection to a FrontBase server. The following defaults are assumed for missing optional parameters: *host* = 'localhost', *username* = "_SYSTEM" and *password* = empty password.

fbsql_pconnect() acts very much like `fbsql_connect()` with two major differences.

To set Frontbase server port number, use `fbsql_select_db()`.

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.

This type of links is therefore called 'persistent'.

fbsql_query (PHP 4 >= 4.0.6)

Send a FrontBase query

resource **fbsql_query** (string query [, resource link_identifier]) \linebreak

fbsql_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 **fbsql_connect()** was called with no arguments, and use it.

Note: The query string shall always end with a semicolon.

fbsql_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 **fbsql_query()** fails and returns FALSE:

Example 1. fbsql_query() example

```
<?php
$result = fbsql_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 **fbsql_query()** fails and returns FALSE:

Example 2. fbsql_query() example

```
<?php
$result = fbsql_query ("SELECT my_col FROM my_tbl")
```

```

    or die ("Invalid query");
?>

```

fbsql_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 `fbsql_num_rows()` to find out how many rows were returned for a `SELECT` statement or `fbsql_affected_rows()` to find out how many rows were affected by a `DELETE`, `INSERT`, `REPLACE`, or `UPDATE` statement.

For `SELECT` statements, **fbsql_query()** returns a new result identifier that you can pass to `fbsql_result()`. When you are done with the result set, you can free the resources associated with it by calling `fbsql_free_result()`. Although, the memory will automatically be freed at the end of the script's execution.

See also: `fbsql_affected_rows()`, `fbsql_db_query()`, `fbsql_free_result()`, `fbsql_result()`, `fbsql_select_db()`, and `fbsql_connect()`.

fbsql_read_blob (PHP 4 >= 4.2.0)

Read a BLOB from the database

```
string fbsql_read_blob ( string blob_handle [, resource link_identifier] ) \linebreak
```

Returns: A string containing the BLOB specified by `blob_handle`.

fbsql_read_blob() reads BLOB data from the database. If a select statement contains BLOB and/or BLOB columns FrontBase will return the data directly when data is fetched. This default behavior can be changed with `fbsql_set_lob_mode()` so the fetch functions will return handles to BLOB and CLOB data. If a handle is fetched a user must call **fbsql_read_blob()** to get the actual BLOB data from the database.

Example 1. fbsql_read_blob() example

```

<?php
    $link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
    $sql = "SELECT BLOB_COLUMN FROM BLOB_TABLE;";
    $rs = fbsql_query($sql, $link);
    $row_data = fbsql_fetch_row($rs);
    // $row_data[0] will now contain the blob data for teh first row
    fbsql_free_result($rs);

    $rs = fbsql_query($sql, $link);
    fbsql_set_lob_mode($rs, FBSQL_LOB_HANDLE);
    $row_data = fbsql_fetch_row($rs);
    // $row_data[0] will now contain a handle to the BLOB data in the first row
    $blob_data = fbsql_read_blob($row_data[0]);
    fbsql_free_result($rs);

```

```
?>
```

See also: `fbsql_create_blob()`, `fbsql_read_blob()`, `fbsql_read_clob()`, and `fbsql_set_lob_mode()`.

fbsql_read_clob (PHP 4 >= 4.2.0)

Read a CLOB from the database

```
string fbsql_read_clob ( string clob_handle [, resource link_identifier] ) \linebreak
```

Returns: A string containing the CLOB specified by `clob_handle`.

fbsql_read_clob() reads CLOB data from the database. If a select statement contains BLOB and/or CLOB columns FrontBase will return the data directly when data is fetched. This default behavior can be changed with `fbsql_set_lob_mode()` so the fetch functions will return handles to BLOB and CLOB data. If a handle is fetched a user must call **fbsql_read_clob()** to get the actual CLOB data from the database.

Example 1. fbsql_read_clob() example

```
<?php
    $link = fbsql_pconnect ("localhost", "_SYSTEM", "secret")
        or die ("Could not connect");
    $sql = "SELECT CLOB_COLUMN FROM CLOB_TABLE;";
    $rs = fbsql_query($sql, $link);
    $row_data = fbsql_fetch_row($rs);
    // $row_data[0] will now contain the clob data for teh first row
    fbsql_free_result($rs);

    $rs = fbsql_query($sql, $link);
    fbsql_set_lob_mode($rs, FBSQL_LOB_HANDLE);
    $row_data = fbsql_fetch_row($rs);
    // $row_data[0] will now contain a handle to the CLOB data in the first row
    $clob_data = fbsql_read_clob($row_data[0]);
    fbsql_free_result($rs);

?>
```

See also: `fbsql_create_blob()`, `fbsql_read_blob()`, `fbsql_read_clob()`, and `fbsql_set_lob_mode()`.

fbsql_result (PHP 4 >= 4.0.6)

Get result data

mixed **fbsql_result** (resource result, int row [, mixed field]) \linebreak

fbsql_result() returns the contents of one cell from a FrontBase 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 **fbsql_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 **fbsql_result()** should not be mixed with calls to other functions that deal with the result set.

Recommended high-performance alternatives: **fbsql_fetch_row()**, **fbsql_fetch_array()**, and **fbsql_fetch_object()**.

fbsql_rollback (PHP 4 >= 4.0.6)

Rollback a transaction to the database

bool **fbsql_rollback** ([resource link_identifier]) \linebreak

Returns: TRUE on success, FALSE on failure.

fbsql_rollback() ends the current transaction by rolling back all statements issued since last commit. This command is only needed if autocommit is set to false.

See also: **fbsql_autocommit()** and **fbsql_commit()**

fbsql_select_db (PHP 4 >= 4.0.6)

Select a FrontBase database

bool **fbsql_select_db** (string database_name [, resource link_identifier]) \linebreak

Returns: TRUE on success, FALSE on error.

fbsql_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 **fbsql_connect()** was called, and use it.

The client contacts FBExec to obtain the port number to use for the connection to the database. If the database name is a number the system will use that as a port number and it will not ask FBExec for the port number. The FrontBase server can be started as `FFrontBase -FBExec=No -port=<port number> <database name>`.

Every subsequent call to `fbsql_query()` will be made on the active database.

if the database is protected with a database password, the user must call `fbsql_database_password()` before selecting the database.

See also: `fbsql_connect()`, `fbsql_pconnect()`, `fbsql_database_password()` and `fbsql_query()`.

fbsql_set_lob_mode (PHP 4 >= 4.2.0)

Set the LOB retrieve mode for a FrontBase result set

```
bool fbsql_set_lob_mode ( resource result, string database_name) \linebreak
```

Returns: `TRUE` on success, `FALSE` on error.

fbsql_set_lob_mode() sets the mode for retrieving LOB data from the database. When BLOB and CLOB data is stored in FrontBase it can be stored direct or indirect. Direct stored LOB data will always be fetched no matter the setting of the lob mode. If the LOB data is less than 512 bytes it will always be stored directly.

- `FBSQL_LOB_DIRECT` - LOB data is retrieved directly. When data is fetched from the database with `fbsql_fetch_row()`, and other fetch functions, all CLOB and BLOB columns will be returned as ordinary columns. This is the default value on a new FrontBase result.
- `FBSQL_LOB_HANDLE` - LOB data is retrieved as handles to the data. When data is fetched from the database with **`fbsql_fetch_row`** (), and other fetch functions, LOB data will be returned as a handle to the data if the data is stored indirect or the data if it is stored direct. If a handle is returned it will be a 27 byte string formatted as "@'00000000000000000000000000000000'".

See also: `fbsql_create_blob()`, `fbsql_create_clob()`, `fbsql_read_blob()`, and `fbsql_read_clob()`.

fbsql_set_transaction (PHP 4 >= 4.2.0)

Set the transaction locking and isolation

```
void fbsql_set_transaction ( resource link_identifier, int Locking, int Isolation) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

fbsql_start_db (PHP 4 >= 4.0.6)

Start a database on local or remote server

```
bool fbsql_start_db ( string database_name [, resource link_identifier]) \linebreak
```

Returns: TRUE on success, FALSE on failure.

fbsql_start_db()

See also: fbsql_db_status() and fbsql_stop_db().

fbsql_stop_db (PHP 4 >= 4.0.6)

Stop a database on local or remote server

```
bool fbsql_stop_db ( string database_name [, resource link_identifier]) \linebreak
```

Returns: TRUE on success, FALSE on failure.

fbsql_stop_db()

See also: fbsql_db_status() and fbsql_start_db().

fbsql_tablename (PHP 4 >= 4.2.0)

Get table name of field

```
string fbsql_tablename ( resource result, int i) \linebreak
```

fbsql_tablename() takes a result pointer returned by the fbsql_list_tables() function as well as an integer index and returns the name of a table. The fbsql_num_rows() function may be used to determine the number of tables in the result pointer.

Example 1. fbsql_tablename() example

```
<?php
fbsql_connect ("localhost", "_SYSTEM", "");
$result = fbsql_list_tables ("wisconsin");
$i = 0;
while ($i < fbsql_num_rows ($result)) {
    $tb_names[$i] = fbsql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

fbsql_username (PHP 4 >= 4.0.6)

Get or set the host user used with a connection

string **fbsql_username** (resource link_identifier [, string username]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

fbsql_warnings (PHP 4 >= 4.0.6)

Enable or disable FrontBase warnings

bool **fbsql_warnings** ([bool OnOff]) \linebreak

Returns TRUE if warnings is turned on otherwise FALSE.

fbsql_warnings() enables or disables FrontBase warnings.

XXIX. filePro functions

These functions allow read-only access to data stored in filePro databases.

filePro is a registered trademark of fP Technologies, Inc. You can find more information about filePro at <http://www.fptech.com/>.

filepro (PHP 3, PHP 4 >= 4.0.0)

Read and verify the map file

```
bool filepro ( string directory) \linebreak
```

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.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

filepro_fieldcount (PHP 3, PHP 4 >= 4.0.0)

Find out how many fields are in a filePro database

```
int filepro_fieldcount ( void) \linebreak
```

Returns the number of fields (columns) in the opened filePro database.

See also filepro().

filepro_fieldname (PHP 3, PHP 4 >= 4.0.0)

Gets the name of a field

```
string filepro_fieldname ( int field_number) \linebreak
```

Returns the name of the field corresponding to *field_number*.

filepro_fieldtype (PHP 3, PHP 4 >= 4.0.0)

Gets the type of a field

```
string filepro_fieldtype ( int field_number) \linebreak
```

Returns the edit type of the field corresponding to *field_number*.

filepro_fieldwidth (PHP 3, PHP 4 >= 4.0.0)

Gets the width of a field

```
int filepro_fieldwidth ( int field_number) \linebreak
```

Returns the width of the field corresponding to *field_number*.

filepro_retrieve (PHP 3, PHP 4 >= 4.0.0)

Retrieves data from a filePro database

```
string filepro_retrieve ( int row_number, int field_number) \linebreak
```

Returns the data from the specified location in the database.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

filepro_rowcount (PHP 3, PHP 4 >= 4.0.0)

Find out how many rows are in a filePro database

```
int filepro_rowcount ( void) \linebreak
```

Returns the number of rows in the opened filePro database.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

See also filepro().

XXX. Filesystem functions

For related functions, see also the Directory and Program Execution sections.

basename (PHP 3, PHP 4 >= 4.0.0)

Returns filename component of path

string **basename** (string path [, string suffix]) \linebreak

Given a string containing a path to a file, this function will return the base name of the file. If the filename ends in *suffix* this will also be cut off.

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.php";
$file = basename ($path);           // $file is set to "index.php"
$file = basename ($path, ".php");   // $file is set to "index"
```

Note: The *suffix* parameter was added in PHP 4.1.0.

See also: `dirname()`

chgrp (PHP 3, PHP 4 >= 4.0.0)

Changes file group

int **chgrp** (string filename, mixed group) \linebreak

Attempts to change the group of the file *filename* to *group* (specified by name or number). 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, FALSE on failure.

See also `chown()` and `chmod()`.

Note: This function is not implemented on Windows platforms.

chmod (PHP 3, PHP 4 >= 4.0.0)

Changes file mode

```
int chmod ( string filename, int mode) \linebreak
```

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+rx,g+rx"); // string; incorrect
chmod ("/somedir/somefile", 0755); // octal; correct value of mode
```

Returns TRUE on success, FALSE on failure.

See also `chown()` and `chgrp()`.

Note: This function is not implemented on Windows platforms.

chown (PHP 3, PHP 4 >= 4.0.0)

Changes file owner

```
int chown ( string filename, mixed user) \linebreak
```

Attempts to change the owner of the file *filename* to user *user* (specified by name or number). Only the superuser may change the owner of a file.

Returns TRUE on success, FALSE on failure.

See also `chown()` and `chmod()`.

Note: This function is not implemented on Windows platforms.

clearstatcache (PHP 3, PHP 4 >= 4.0.0)

Clears file stat cache

void **clearstatcache** (void) \linebreak

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 >= 4.0.0)

Copies file

int **copy** (string source, string dest) \linebreak

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");
}
```

Note: As of PHP 4.3.0, both *source* and *dest* may be URLs if the "fopen wrappers" have been enabled. See fopen() for more details.

Warning

If the destination file already exists, it will be overwritten.

See also move_uploaded_file(), rename(), and the section of the manual about handling file uploads.

delete (unknown)

A dummy manual entry

void **delete** (string file) \linebreak

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 >= 4.0.0)

Returns directory name component of path

string **dirname** (string path) \linebreak

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"
```

Note: In PHP 4.0.3, **dirname()** was fixed to be POSIX-compliant. Essentially, this means that if there are no slashes in *path*, a dot ('.') is returned, indicating the current directory. Otherwise, the returned string is *path* with any trailing */component* removed. Note that this means that you will often get a slash or a dot back from **dirname()** in situations where the older functionality would have given you the empty string.

See also: `basename()`

disk_free_space (PHP 4 >= 4.1.0)

Returns available space in directory

float **disk_free_space** (string directory) \linebreak

Given a string containing a directory, this function will return the number of bytes available on the corresponding filesystem or disk partition.

Example 1. disk_free_space() example

```
$df = disk_free_space("/"); // $df contains the number of bytes
                             // available on "/"
```

disk_total_space (PHP 4 >= 4.1.0)

Returns the total size of a directory

```
float disk_total_space ( string directory) \linebreak
```

Given a string containing a directory, this function will return the total number of bytes on the corresponding filesystem or disk partition.

Example 1. disk_total_space() example

```
$df = disk_total_space("/"); // $df contains the total number of
                             // bytes available on "/"
```

diskfreespace (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Alias of disk_free_space()

```
float diskfreespace ( string directory) \linebreak
```

This is a deprecated alias of disk_free_space(). Use that function instead.

fclose (PHP 3, PHP 4 >= 4.0.0)

Closes an open file pointer

```
bool fclose ( int fp) \linebreak
```

The file pointed to by *FP* is closed.

Returns `TRUE` on success, `FALSE` on failure.

The file pointer must be valid, and must point to a file successfully opened by `fopen()` or `fsockopen()`.

fEOF (PHP 3, PHP 4 >= 4.0.0)

Tests for end-of-file on a file pointer

`int fEOF (int fp) \linebreak`

Returns `TRUE` if the file pointer is at EOF or an error occurs; otherwise returns `FALSE`.

The file pointer must be valid, and must point to a file successfully opened by `fopen()`, `popen()`, or `fsockopen()`.

fFlush (PHP 4)

Flushes the output to a file

`int fFlush (int fp) \linebreak`

This function forces a write of all buffered output to the 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 `fsockopen()`.

fgetc (PHP 3, PHP 4 >= 4.0.0)

Gets character from file pointer

`string fgetc (int fp) \linebreak`

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 `fsockopen()`.

See also `fread()`, `fopen()`, `popen()`, `fsockopen()`, and `fgets()`.

fgetcsv (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Gets line from file pointer and parse for CSV fields

array **fgetcsv** (int *fp*, int *length* [, string *delimiter*]) \linebreak

Similar to `fgets()` except that **fgetcsv()** parses the line it reads for fields in CSV format and returns an array containing the fields read. The field delimiter is a comma, unless you specify another delimiter with the optional third parameter.

FP must be a valid file pointer to a file successfully opened by `fopen()`, `popen()`, or `fsockopen()`

Length must be greater than the longest line to be found in the CSV file (allowing for trailing line-end characters).

fgetcsv() returns `FALSE` on error, including end of file.

Note: 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 >= 4.0.0)

Gets line from file pointer

string **fgets** (int *fp* [, int *length*]) \linebreak

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 no `length` is specified, the `length` defaults to 1k, or 1024 bytes.

If an error occurs, returns `FALSE`.

Common Pitfalls:

People used to the 'C' semantics of **fgets()** should note the difference in how `EOF` is returned.

The file pointer must be valid, and must point to a file successfully opened by `fopen()`, `popen()`, or `fsockopen()`.

A simple example follows:

Example 1. Reading a file line by line

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while (!feof ($fd)) {
    $buffer = fgets($fd, 4096);
    echo $buffer;
}
fclose ($fd);
```

Note: The *length* parameter became optional in PHP 4.2.0

See also `fread()`, `fopen()`, `popen()`, `fgetc()`, `fsockopen()`, and `socket_set_timeout()`.

fgetss (PHP 3, PHP 4 >= 4.0.0)

Gets line from file pointer and strip HTML tags

string **fgetss** (int fp, int length [, string allowable_tags]) \linebreak

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, PHP 4.0.0.

See also `fgets()`, `fopen()`, `fsockopen()`, `popen()`, and `strip_tags()`.

file (PHP 3, PHP 4 >= 4.0.0)

Reads entire file into an array

array **file** (string filename [, int use_include_path]) \linebreak

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.

Note: Each line in the resulting array will include the line ending, so you still need to use trim() if you do not want the line ending present.

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
$fcontents = file ('http://www.example.com/');
while (list ($line_num, $line) = each ($fcontents)) {
    echo "<b>Line $line_num:</b>; ", htmlspecialchars ($line), "<br>\n";
}

// get a web page into a string
$fcontents = implode ("", file ('http://www.example.com/'));
?>
```

Note: As of PHP 4.3.0 you can use file_get_contents() to return the contents of a file as a string in a binary safe manner.

Warning

This function is not (yet) binary safe!

Tip: You can use a URL as a filename with this function if the "fopen wrappers" have been enabled. See fopen() for more details.

See also readfile(), fopen(), fsockopen(), and popen().

file_exists (PHP 3, PHP 4 >= 4.0.0)

Checks whether a file exists

bool **file_exists** (string filename) \linebreak

Returns TRUE if the file specified by *filename* exists; FALSE otherwise.

This function 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.

file_get_contents (PHP 4 CVS only)

Reads entire file into a string

string **file_get_contents** (string filename [, int use_include_path]) \linebreak

Identical to `readfile()`, except that **file_get_contents()** returns the file in a string.

Note: This function was introduced in PHP 4.3.0.

Note: This function is binary-safe.

Tip: You can use a URL as a filename with this function if the "fopen wrappers" have been enabled. See `fopen()` for more details.

See also: `fgets()`, `file()`, `fread()`, `include()`, and `readfile()`.

file_get_wrapper_data (PHP 4 CVS only)

Retrieves header/meta data from "wrapped" file pointers

mixed **file_get_wrapper_data** (int fp) \linebreak

This function returns header or meta data from files opened with `fopen()`. This is useful to return the response headers for HTTP connections, or some other statistics for other resources.

The format of the returned data is deliberately undocumented at this time, and depends on which wrapper(s) were used to open the file.

Note: This function was introduced in PHP 4.3.0.

file_register_wrapper (PHP 4 CVS only)

Register a URL wrapper implemented as a PHP class

boolean **file_register_wrapper** (string protocol, string classname) \linebreak

This function is currently only documented by the example below:

Example 1. Implementing a base64 encoding protocol

```
class Base64EncodingStream {
    var $fp = null;

    function stream_open($path, $mode, $options, &$opened_path)
    {
        $this->fp = fopen($path, $mode);
        return is_resource($this->fp);
    }
    function stream_close()
    {
        fclose($this->fp);
    }
    function stream_read($count)
    {
        return false; // We only allow writing
    }
    function stream_write($data)
    {
        return fwrite($this->fp, base64_encode($data));
    }
    function stream_flush()
    {
        fflush($this->fp);
        return true;
    }
    function stream_seek($offset, $whence)
    {
        return false;
    }
    function stream_gets()
    {
        return false;
    }
    function stream_tell()
    {
        return false;
    }
    function stream_eof()
    {
        return false;
    }
}
file_register_wrapper("base64", "Base64EncodingStream")
    or die("Failed to register protocol");

copy("/tmp/inputfile.txt", "base64:///tmp/outputfile.txt");
readfile("/tmp/outputfile");
```

file_register_wrapper() will return false if the *protocol* already has a handler, or if "fopen wrappers" are disabled.

Note: This function was introduced in PHP 4.3.0.

fileatime (PHP 3, PHP 4 >= 4.0.0)

Gets last access time of file

int **fileatime** (string filename) \linebreak

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

filectime (PHP 3, PHP 4 >= 4.0.0)

Gets inode change time of file

int **filectime** (string filename) \linebreak

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 filesystems, a file is considered changed when its inode data is changed; that is, when the permissions, owner, group, or other metadata from the inode is updated. See also `filemtime()` (which is what you want to use when you want to create "Last Modified" footers on web pages) and `fileatime()`.

Note also that in some Unix texts the `ctime` of a file is referred to as being the creation time of the file. This is wrong. There is no creation time for Unix files in most Unix filesystems.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

filegroup (PHP 3, PHP 4 >= 4.0.0)

Gets file group

int **filegroup** (string filename) \linebreak

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 is not implemented on Windows platforms.

Note: This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

fileinode (PHP 3, PHP 4 >= 4.0.0)

Gets file inode

int **fileinode** (string filename) \linebreak

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

Note: This function is not implemented on Windows platforms.

filemtime (PHP 3, PHP 4 >= 4.0.0)

Gets file modification time

int **filemtime** (string filename) \linebreak

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

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.

See also `filectime()` and `touch()`.

fileowner (PHP 3, PHP 4 >= 4.0.0)

Gets file owner

`int fileowner (string filename) \linebreak`

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

Note: This function is not implemented on Windows platforms.

fileperms (PHP 3, PHP 4 >= 4.0.0)

Gets file permissions

`int fileperms (string filename) \linebreak`

Returns the permissions on the file, or `FALSE` in case of an error.

This function 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.

filesize (PHP 3, PHP 4 >= 4.0.0)

Gets file size

int **filesize** (string filename) \linebreak

Returns the size of the file in bytes, or `FALSE` in case of an error.

The results of this function are cached. See `clearstatcache()` for more details.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

filetype (PHP 3, PHP 4 >= 4.0.0)

Gets file type

string **filetype** (string filename) \linebreak

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

See also: `is_dir()`, `is_file()`, `is_link()`, `file_exists()`, and `stat()`.

flock (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Portable advisory file locking

bool **flock** (int fp, int operation [, int wouldblock]) \linebreak

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 Unix derivatives and even Windows). The optional third 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).

Note: Because **flock()** requires a file pointer, you may have to use a special lock file to protect access to a file that you intend to truncate by opening it in write mode (with a "w" or "w+" argument to **fopen()**).

Warning

flock() will not work on NFS and many other networked file systems. Check your operating system documentation for more details.

On some operating systems **flock()** is implemented at the process level. When using a multithreaded server API like ISAPI you may not be able to rely on **flock()** to protect files against other PHP scripts running in parallel threads of the same server instance!

fopen (PHP 3, PHP 4 >= 4.0.0)

Opens file or URL

int **fopen** (string filename, string mode [, int use_include_path]) \linebreak

If *filename* begins with "http://" (not case sensitive), an HTTP 1.0 connection is opened to the specified server, the page is requested using the HTTP GET method, and a file pointer is returned to the beginning of the body of the response. A 'Host:' header is sent with the request in order to handle name-based virtual hosts.

As of PHP 4.3.0 (not yet released), if you have compiled in support for OpenSSL, you may use "https://" to open an HTTP connection over SSL.

Note that the file pointer allows you to retrieve only the *body* of the response; to retrieve the HTTP response header you need to be using PHP 4.0.5 or later; The headers will be stored in the \$http_response_header variable. As of PHP 4.3.0 (not yet released), the header information can be retrieved using the **file_get_wrapper_data()**.

HTTP connections are read-only; you cannot write data or copy files to an HTTP resource.

Versions prior to PHP 4.0.5 do not handle HTTP redirects. Because of this, directories must include trailing slashes.

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 the remote file already exists on the ftp server and you attempt to open it for writing, this will fail. If you need to update existing files over ftp, use ftp_connect().

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.

Note: The `mode` may contain the letter `'b'`. This is useful only on systems which differentiate between binary and text files (i.e. Windows. 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.example.com/", "r");  
$fp = fopen ("ftp://user:password@example.com/", "w");
```

If you are experiencing problems with reading and writing to files and you're using the server module version of PHP, remember to make sure that the files and directories you're using are accessible to the server process.

On the Windows platform, be careful to escape any backslashes used in the path to the file, or use forward slashes.

```
$fp = fopen ("c:\\data\\info.txt", "r");
```

See also `fclose()`, `fsockopen()`, `socket_set_timeout()`, and `popen()`.

fpasssthru (PHP 3, PHP 4 >= 4.0.0)

Output all remaining data on a file pointer

`int fpasssthru (int fp) \linebreak`

Reads to EOF on the given file pointer from the current position and writes the results to standard output.

If an error occurs, **fpasssthru()** returns `FALSE`.

The file pointer must be valid, and must point to a file successfully opened by `fopen()`, `popen()`, or `fsockopen()`. You may need to call `rewind()` to reset the file pointer to the beginning of the file if you have already written data to the file. The file is closed when **fpasssthru()** 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.

Note: When using **fpasssthru()** on a binary file on Windows systems, you should make sure to open the file in binary mode by appending a `b` to the mode used in the call to `fopen()`.

See also `readfile()`, `fopen()`, `popen()`, and `fsockopen()`

fputs (PHP 3, PHP 4 >= 4.0.0)

Writes to a file pointer

`int fputs (int fp, string str [, int length]) \linebreak`

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 >= 4.0.0)

Binary-safe file read

`string fread (int fp, int length) \linebreak`

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");
$content = fread ($fd, filesize ($filename));
fclose ($fd);
```

Note: On systems which differentiate between binary and text files (i.e. Windows) the file must be opened with 'b' included in fopen() mode parameter.

```
$filename = "c:\\files\\somepic.gif";
$fd = fopen ($filename, "rb");
$content = fread ($fd, filesize ($filename));
fclose ($fd);
```

See also fwrite(), fopen(), fsockopen(), popen(), fgets(), fgetss(), fscanf(), file(), and fpassthru().

fscanf (PHP 4)

Parses input from a file according to a format

mixed **fscanf** (int handle, string format [, string var1]) \linebreak

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()`, `scanf()`, `printf()`, and `sprintf()`.

fseek (PHP 3, PHP 4 >= 4.0.0)

Seeks on a file pointer

int **fseek** (int *fp*, int *offset* [, int *whence*]) \linebreak

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*. (To move to a position before the end-of-file, you need to pass a negative *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 4.0.0.

See also `ftell()` and `rewind()`.

fstat (PHP 4 >= 4.0.0)

Gets information about a file using an open file pointer

array **fstat** (int *fp*) \linebreak

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.

ftell (PHP 3, PHP 4 >= 4.0.0)

Tells file pointer read/write position

`int ftell (int fp) \linebreak`

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 >= 4.0.0)

Truncates a file to a given length

`int ftruncate (int fp, int size) \linebreak`

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 >= 4.0.0)

Binary-safe file write

int **fwrite** (int *fp*, string *string* [, int *length*]) \linebreak

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.

fwrite() returns the number of bytes written, or -1 on error.

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*.

Note: On systems which differentiate between binary and text files (i.e. Windows) the file must be opened with 'b' included in `fopen()` mode parameter.

See also `fread()`, `fopen()`, `fsockopen()`, `popen()`, and `fputs()`.

is_dir (PHP 3, PHP 4 >= 4.0.0)

Tells whether the filename is a directory

bool **is_dir** (string *filename*) \linebreak

Returns `TRUE` if the filename exists and is a directory. If *filename* is a relative filename, it will be checked relative to the current working directory.

The results of this function are cached. See `clearstatcache()` for more details.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

See also `chdir()`, `is_file()` and `is_link()`.

is_executable (PHP 3, PHP 4 >= 4.0.0)

Tells whether the filename is executable

bool **is_executable** (string *filename*) \linebreak

Returns `TRUE` if the filename exists and is executable.

The results of this function are cached. See `clearstatcache()` for more details.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

See also `is_file()` and `is_link()`.

is_file (PHP 3, PHP 4 >= 4.0.0)

Tells whether the filename is a regular file

`bool is_file (string filename) \linebreak`

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 >= 4.0.0)

Tells whether the filename is a symbolic link

`bool is_link (string filename) \linebreak`

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()`, `is_file()`, and `readlink()`.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

Note: This function is not implemented on Windows platforms.

is_readable (PHP 3, PHP 4 >= 4.0.0)

Tells whether the filename is readable

`bool is_readable (string filename) \linebreak`

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

See also `is_writable()`.

`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) \linebreak
```

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.

`is_uploaded_file()` is available only in versions of PHP 3 after PHP 3.0.16, and in versions of PHP 4 after 4.0.2. If you are stuck using an earlier version, you can use the following function to help protect yourself:

Note: The following example 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);
}

/* This is how to use it, since you also don't have
 * move_uploaded_file() in these older versions: */
if (is_uploaded_file($_HTTP_POST_FILES['userfile'])) {
    copy($_HTTP_POST_FILES['userfile'], "/place/to/put/uploaded/file");
} else {
    echo "Possible file upload attack: filename '$_HTTP_POST_FILES[userfile]'.";
}
?>
```

See also `move_uploaded_file()`, and the section Handling file uploads for a simple usage example.

is_writable (PHP 4 >= 4.0.0)

Tells whether the filename is writable

bool **is_writable** (string filename) \linebreak

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.

This function will not work on remote files; the file to be examined must be accessible via the server's filesystem.

See also `is_readable()`.

is_writeable (PHP 3, PHP 4 >= 4.0.0)

Tells whether the filename is writable

bool **is_writeable** (string filename) \linebreak

This is an alias for `is_writable()`

link (PHP 3, PHP 4 >= 4.0.0)

Create a hard link

int **link** (string target, string link) \linebreak

link() creates a hard link.

See also the `symlink()` to create soft links, and `readlink()` along with `linkinfo()`.

Note: This function is not implemented on Windows platforms.

linkinfo (PHP 3, PHP 4 >= 4.0.0)

Gets information about a link

int **linkinfo** (string path) \linebreak

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 is not implemented on Windows platforms.

lstat (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Gives information about a file or symbolic link

array **lstat** (string filename) \linebreak

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.

lstat() cannot be used on remote files.

The results of this function are cached. See `clearstatcache()` for more details.

mkdir (PHP 3, PHP 4 >= 4.0.0)

Makes directory

```
int mkdir ( string pathname, int mode) \linebreak
```

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. The mode is also modified by the current `umask`, which you can change using `umask()`.

```
mkdir ("/path/to/my/dir", 0700);
```

Returns `TRUE` on success, `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) \linebreak
```

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.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

Note: **move_uploaded_file()** is not affected by the normal safe-mode UID-restrictions. This is not unsafe because **move_uploaded_file()** only operates on files uploaded via PHP.

Warning

If the destination file already exists, it will be overwritten.

See also `is_uploaded_file()`, and the section Handling file uploads for a simple usage example.

parse_ini_file (PHP 4 >= 4.0.0)

Parse a configuration file

array **parse_ini_file** (string filename [, bool process_sections]) \linebreak

parse_ini_file() loads in the ini file specified in *filename*, and returns the settings in it in an associative array. By setting the last *process_sections* parameter to `TRUE`, you get a multidimensional array, with the section names and settings included. The default for *process_sections* is `FALSE`

Note: This function has nothing to do with the `php.ini` file. It is already processed, the time you run your script. This function can be used to read in your own application's configuration files.

The structure of the ini file is similar to that of the `php.ini`'s.

Example 1. Contents of sample.ini

```
; This is a sample configuration file
; Comments start with ';', as in php.ini

[first_section]
one = 1
five = 5

[second_section]
path = /usr/local/bin
```

Example 2. parse_ini_file() example

```
<?php

// Parse without sections
$ini_array = parse_ini_file("sample.ini");
```

```

print_r($ini_array);

// Parse with sections
$ini_array = parse_ini_file("sample.ini", TRUE);
print_r($ini_array);

?>

```

Would produce:

```

Array
(
    [one] => 1
    [five] => 5
    [path] => /usr/local/bin
)
Array
(
    [first_section] => Array
        (
            [one] => 1
            [five] => 5
        )

    [second_section] => Array
        (
            [path] => /usr/local/bin
        )
)

```

pathinfo (PHP 4 >= 4.0.3)

Returns information about a file path

array **pathinfo** (string path) \linebreak

pathinfo() returns an associative array containing information about *path*. The following array elements are returned: *dirname*, *basename* and *extension*.

Example 1. pathinfo() Example

```
<?php

$path_parts = pathinfo("/www/htdocs/index.html");

echo $path_parts["dirname"] . "\n";
echo $path_parts["basename"] . "\n";
echo $path_parts["extension"] . "\n";

?>
```

Would produce:

```
/www/htdocs
index.html
html
```

See also `dirname()`, `basename()`, `parse_url()` and `realpath()`.

pclose (PHP 3, PHP 4 >= 4.0.0)

Closes process file pointer

int pclose (int fp) \linebreak

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 >= 4.0.0)

Opens process file pointer

int popen (string command, string mode) \linebreak

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");
```

Note: If the command to be executed could not be found, a valid resource is returned. This may seem odd, but makes sense; it allows you to access any error message returned by the shell:

```
<?php
error_reporting(E_ALL);

/* Add redirection so we can get stderr. */
$fp = popen('/path/to/spooge 2>&l', 'r');
echo "'$fp'; " . gettype($fp) . "\n";
$read = fread($fp, 2096);
echo $read;
pclose($fp);
?>
```

See also `pclose()`.

readfile (PHP 3, PHP 4 >= 4.0.0)

Outputs a file

`int readfile (string filename [, int use_include_path]) \linebreak`

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.

Versions prior to PHP 4.0.5 do not handle HTTP redirects. Because of this, directories must include trailing slashes.

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 >= 4.0.0)

Returns the target of a symbolic link

string **readlink** (string path) \linebreak

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 `is_link()`, `symlink()`, and `linkinfo()`.

Note: This function is not implemented on Windows platforms.

realpath (PHP 4 >= 4.0.0)

Returns canonicalized absolute pathname

string **realpath** (string path) \linebreak

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.

realpath() returns `FALSE` on failure, e.g. if the file does not exist.

Example 1. realpath() example

```
$real_path = realpath ( "../..../index.php" );
```


See also: `basename()`, `dirname()`, and `path_info()`.

rename (PHP 3, PHP 4 >= 4.0.0)

Renames a file

`int rename (string oldname, string newname) \linebreak`

Attempts to rename *oldname* to *newname*.

Returns `TRUE` on success, `FALSE` on failure.

rewind (PHP 3, PHP 4 >= 4.0.0)

Rewind the position of a file pointer

`int rewind (int fp) \linebreak`

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()`.

Note: If you have opened the file in append ("a") mode, any data you write to the file will always be appended, regardless of the file position.

See also `fseek()` and `ftell()`.

rmdir (PHP 3, PHP 4 >= 4.0.0)

Removes directory

`bool rmdir (string dirname) \linebreak`

Attempts to remove the directory named by *dirname*. The directory must be empty, and the relevant permissions must permit this. Returns `TRUE` on success, `FALSE` on failure.

See also `mkdir()` and `unlink()`.

set_file_buffer (PHP 3 >= 3.0.8, PHP 4)

Sets file buffering on the given file pointer

`int set_file_buffer (int fp, int buffer) \linebreak`

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()`.

stat (PHP 3, PHP 4 >= 4.0.0)

Gives information about a file

`array stat (string filename) \linebreak`

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.

Returns `FALSE` in case of error.

stat() cannot be used on remote files.

The results of this function are cached. See `clearstatcache()` for more details.

symlink (PHP 3, PHP 4 >= 4.0.0)

Creates a symbolic link

int **symlink** (string *target*, string *link*) \linebreak

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 is not implemented on Windows platforms.

tempnam (PHP 3, PHP 4 >= 4.0.0)

Create file with unique file name

string **tempnam** (string *dir*, string *prefix*) \linebreak

Creates a file with a unique filename in the specified directory. If the directory does not exist, **tempnam()** may generate a file in the system's temporary directory, and return the name of that.

Prior to PHP 4.0.6, the behaviour of the **tempnam()** function was 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 `FALSE` string on failure.

Example 1. tempnam() example

```

$tmpfname = tempnam ( "/tmp", "FOO" );

$fp = fopen($tmpfname, "w");
fwrite($fp, "writing to tempfile");
fclose($fp);

// do here something

unlink($tmpfname);

```

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 the script gets around to creating the file. Note, that you need to remove the file in case you need it no more, it is not done automatically.

See also `tmpfile()` and `unlink()`.

tmpfile (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Creates a temporary file

int **tmpfile** (void) \linebreak

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.

Example 1. tmpfile() example

```

$tmp = tmpfile();
fwrite($tmp, "writing to tempfile");
fclose($tmp); // this removes the file

```

See also `tempnam()`.

touch (PHP 3, PHP 4 >= 4.0.0)

Sets access and modification time of file

int **touch** (string filename [, int time]) \linebreak

Attempts to set the access and 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. This is equivalent to what utime (sometimes referred to as utimes) does.

If the file does not exist, it is created.

Returns TRUE on success, FALSE on failure.

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 >= 4.0.0)

Changes the current umask

int **umask** (int mask) \linebreak

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 is not implemented on Windows platforms.

unlink (PHP 3, PHP 4 >= 4.0.0)

Deletes a file

int **unlink** (string filename) \linebreak

Deletes *filename*. Similar to the Unix C `unlink()` function.

Returns `TRUE` on success, `FALSE` on failure.

See also `rmdir()` for removing directories.

XXXI. 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_add_template (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Adds a template into the FDF document

```
bool fdf_add_template ( int fdfdoc, int newpage, string filename, string template, int rename) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

fdf_close (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Close an FDF document

```
bool fdf_close ( int fdf_document) \linebreak
```

The **fdf_close()** function closes the FDF document.

See also `fdf_open()`.

fdf_create (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Create a new FDF document

```
int fdf_create ( void) \linebreak
```

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_get_file (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the value of the /F key

string **fdf_get_file** (int fdf_document) \linebreak

The `fdf_set_file()` returns the value of the /F key.

See also `fdf_set_file()`.

fdf_get_status (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the value of the /STATUS key

string **fdf_get_status** (int fdf_document) \linebreak

The `fdf_get_status()` returns the value of the /STATUS key.

See also `fdf_set_status()`.

fdf_get_value (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the value of a field

string **fdf_get_value** (int fdf_document, string fieldname) \linebreak

The `fdf_get_value()` function returns the value of a field.

See also `fdf_set_value()`.

fdf_next_field_name (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the next field name

string **fdf_next_field_name** (int fdf_document [, string fieldname]) \linebreak

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_value()`, `fdf_get_value()`.

fdf_open (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Open a FDF document

int **fdf_open** (string filename) \linebreak

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_save (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Save a FDF document

int **fdf_save** (string filename) \linebreak

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_set_ap (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Set the appearance of a field

```
bool fdf_set_ap ( int fdf_document, string field_name, int face, string filename, int page_number) \linebreak
```

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=FDNormalAP, 2=FDRollerAP, 3=FDDownAP.

fdf_set_encoding (PHP 4 >= 4.1.0)

Sets FDF character encoding

```
bool fdf_set_encoding ( int fdf_document, string encoding) \linebreak
```

fdf_set_encoding() sets the character encoding in FDF document *fdf_document*. *encoding* should be the valid encoding name. The valid encoding name in Acrobat 5.0 are "Shift-JIS", "UHC", "GBK", "BigFive".

The **fdf_set_encoding()** is available in PHP 4.1.0 or later.

fdf_set_file (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Set the value of the /F key

```
bool fdf_set_file ( int fdf_document, string filename) \linebreak
```

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_set_flags (PHP 4 >= 4.0.2)

Sets a flag of a field

```
bool fdf_set_flags ( int fdf_document, string fieldname, int whichFlags, int newFlags) \linebreak
```

The **fdf_set_flags()** sets certain flags of the given field *fieldname*.

See also **fdf_set_opt()**.

fdf_set_javascript_action (PHP 4 >= 4.0.2)

Sets an javascript action of a field

bool **fdf_set_javascript_action** (int fdf_document, string fieldname, int trigger, string script) \linebreak

fdf_set_javascript_action() sets a javascript action for the given field *fieldname*.

See also fdf_set_submit_form_action().

fdf_set_opt (PHP 4 >= 4.0.2)

Sets an option of a field

bool **fdf_set_opt** (int fdf_document, string fieldname, int element, string str1, string str2) \linebreak

The **fdf_set_opt()** sets options of the given field *fieldname*.

See also fdf_set_flags().

fdf_set_status (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Set the value of the /STATUS key

bool **fdf_set_status** (int fdf_document, string status) \linebreak

The **fdf_set_status()** sets the value of the /STATUS key.

See also fdf_get_status().

fdf_set_submit_form_action (PHP 4 >= 4.0.2)

Sets a submit form action of a field

bool **fdf_set_submit_form_action** (int fdf_document, string fieldname, int trigger, string script, int flags) \linebreak

The **fdf_set_submit_form_action()** sets a submit form action for the given field *fieldname*.

See also fdf_set_javascript_action().

fdf_set_value (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Set the value of a field

```
bool fdf_set_value ( int fdf_document, string fieldname, string value, int isName) \linebreak
```

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()`.

XXXII. FriBiDi functions

fribidi_log2vis (PHP 4 >= 4.0.4)

Convert a logical string to a visual one

string **fribidi_log2vis** (string str, string direction, int charset) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

XXXIII. FTP functions

The functions in this extension implement client access to file servers speaking the File Transfer Protocol FTP as defined in <http://www.faqs.org/rfcs/rfc959.html>.

Requirements

The FTP-extension has no special requirements. It's completely contained in PHP.

Installation

In order to use FTP functions with your PHP configuration, you should add the `--enable-ftp` option when installing PHP 4, and `--with-ftp` when using PHP 3.

Configuration

This extension does not define any configuration directives.

Resource types

This extension uses one resource-type, which is the link-identifier of the ftp-connection.

Constants

The following constants are defined when using the FTP module: `FTP_ASCII` and `FTP_BINARY`.

Example

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 $ftp_user_name";
    die;
} else {
    echo "Connected to $ftp_server, for user $ftp_user_name";
}

// 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_close($conn_id);
?>
```

ftp_cdup (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Changes to the parent directory

bool **ftp_cdup** (resource ftp_stream) \linebreak

Changes to the parent directory.

Returns TRUE on success, FALSE on error.

ftp_chdir (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Changes directories on a FTP server

bool **ftp_chdir** (resource ftp_stream, string directory) \linebreak

Changes to the specified *directory*.

Returns TRUE on success, FALSE on error.

ftp_close (PHP 4 >= 4.2.0)

Closes an FTP connection

void **ftp_close** (resource ftp_stream) \linebreak

Note: This function is only available in CVS.

ftp_close() closes *ftp_stream* and releases the resource. You can't reuse this resource but have to create a new one with **ftp_connect()**.

ftp_connect (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Opens up an FTP connection

resource **ftp_connect** (string host [, int port [, int timeout]]) \linebreak

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.

The *timeout* parameter specifies the timeout for all subsequent network operations. If omitted, the default value is 90 seconds. The timeout can be changed and queried anytime with **ftp_set_option()** and **ftp_get_option()**.

Note: This parameter is only available in CVS.

ftp_delete (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Deletes a file on the FTP server

bool **ftp_delete** (resource ftp_stream, string path) \linebreak

ftp_delete() deletes the file specified by *path* from the FTP server.

Returns TRUE on success, FALSE on error.

ftp_exec (PHP 4 >= 4.0.3)

Request execution of a program on the FTP server

bool **ftp_exec** (resource stream, string command) \linebreak

Sends a SITE EXEC *command* request to the FTP server. Returns FALSE if the request fails, returns the output of the command otherwise.

ftp_fget (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Downloads a file from the FTP server and saves to an open file

bool **ftp_fget** (resource ftp_stream, resource fp, string remote_file, int mode) \linebreak

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.

Returns TRUE on success, FALSE on error.

ftp_fput (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Uploads from an open file to the FTP server

bool **ftp_fput** (resource ftp_stream, string remote_file, resource fp, int mode) \linebreak

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`

Returns `TRUE` on success, `FALSE` on error.

ftp_get (PHP 3 >= 3.0.13, PHP 4 >= 4.0.0)

Downloads a file from the FTP server

bool **ftp_get** (resource ftp_stream, string local_file, string remote_file, int mode) \linebreak

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`.

Returns `TRUE` on success, `FALSE` on error.

See also `ftp_fget()`.

ftp_get_option (PHP 4 >= 4.2.0)

Retrieves various runtime behaviours of the current FTP stream

mixed **ftp_get_option** (resource stream, int option) \linebreak

Note: This function is only available in CVS.

Returns the value on success, or `FALSE` if the given *option* is not supported. In the latter case, a warning message is also thrown.

This function returns the value for the requested *option* from the specified *stream* . Currently, the following options are supported:

Table 1. Supported runtime FTP options

FTP_TIMEOUT_SEC	Returns the current timeout used for network related operations.
-----------------	--

Example 1. ftp_get_option() example

```
// Get the timeout of the given FTP stream
$timeout = ftp_get_option($conn_id, FTP_TIMEOUT_SEC);
```

ftp_login (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Logs in an FTP connection

bool **ftp_login** (resource ftp_stream, string username, string password) \linebreak

Logs in the given FTP stream.

Returns TRUE on success, FALSE on error.

ftp_mdtm (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the last modified time of the given file

int **ftp_mdtm** (resource ftp_stream, string remote_file) \linebreak

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.

Returns a UNIX timestamp on success, or -1 on error.

Note: **ftp_mdtm()** does not work with directories.

ftp_mkdir (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Creates a directory

string **ftp_mkdir** (resource ftp_stream, string directory) \linebreak

Creates the specified *directory*.

Returns the newly created directory name on success, FALSE on error.

ftp_nlist (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns a list of files in the given directory

array **ftp_nlist** (resource ftp_stream, string directory) \linebreak

Returns an array of filenames from the specified directory on success, `FALSE` on error.

ftp_pasv (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Turns passive mode on or off

bool **ftp_pasv** (resource ftp_stream, bool pasv) \linebreak

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.

Returns `TRUE` on success, `FALSE` on error.

ftp_put (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Uploads a file to the FTP server

bool **ftp_put** (resource ftp_stream, string remote_file, string local_file, int mode) \linebreak

ftp_put() stores *local_file* on the FTP server, as *remote_file*. The transfer *mode* specified must be either `FTP_ASCII` or `FTP_BINARY`.

Returns `TRUE` on success, `FALSE` on error.

Example 1. ftp_put() example

```
$upload = ftp_put($conn_id, $destination_file, $source_file, FTP_ASCII);
```

ftp_pwd (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the current directory name

string **ftp_pwd** (resource ftp_stream) \linebreak

Returns the current directory, or `FALSE` on error.

ftp_quit (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Closes an FTP connection

void **ftp_quit** (resource ftp_stream) \linebreak

ftp_quit() is an alias for **ftp_close()**.

ftp_rawlist (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns a detailed list of files in the given directory

array **ftp_rawlist** (resource ftp_stream, string directory) \linebreak

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_rename (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Renames a file on the FTP server

bool **ftp_rename** (resource ftp_stream, string from, string to) \linebreak

ftp_rename() renames the file or directory currently named *from* to the new name *to*, on the FTP stream *ftp_stream*.

Returns TRUE on success, FALSE on error.

ftp_rmdir (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Removes a directory

bool **ftp_rmdir** (resource ftp_stream, string directory) \linebreak

Removes the specified *directory*.

Returns TRUE on success, FALSE on error.

ftp_set_option (PHP 4 >= 4.2.0)

Set miscellaneous runtime FTP options

bool **ftp_set_option** (resource stream, int option, mixed value) \linebreak

Note: This function is only available in CVS.

Returns `TRUE` if the option could be set; `FALSE` if not. A warning message will be thrown if the *option* is not supported or the passed *value* doesn't match the expected value for the given *option*.

This function controls various runtime options for the specified FTP stream. The *value* parameter depends on which *option* parameter is chosen to be altered. Currently, the following options are supported:

Table 1. Supported runtime FTP options

FTP_TIMEOUT_SEC	Changes the timeout in seconds used for all network related functions. Parameter <i>value</i> has to be of type int and must be greater than 0. The default timeout is 90 seconds.
-----------------	--

Example 1. ftp_set_option() example

```
// Set the network timeout down to 10 seconds
ftp_set_option($conn_id, FTP_TIMEOUT_SEC, 10);
```

ftp_site (PHP 3 >= 3.0.15, PHP 4 >= 4.0.0)

Sends a SITE command to the server

bool **ftp_site** (resource ftp_stream, string cmd) \linebreak

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.

Returns `TRUE` on success, `FALSE` on error.

ftp_size (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the size of the given file

int **ftp_size** (resource ftp_stream, string remote_file) \linebreak

ftp_size() returns the size of a file in bytes. If an error occurs, or if the file does not exist, -1 is returned. Not all servers support this feature.

Returns the file size on success, or -1 on error.

ftp_systype (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the system type identifier of the remote FTP server

string **ftp_systype** (resource ftp_stream) \linebreak

Returns the remote system type, or FALSE on error.

XXXIV. Function Handling functions

These functions all handle various operations involved in working with functions.

call_user_func (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Call a user function given by the first parameter

mixed **call_user_func** (string *function_name* [, mixed *parameter* [, mixed ...]]) \linebreak

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");
```

See also: `call_user_func_array()`, `call_user_method()`, `call_user_method_array()`.

call_user_func_array (PHP 4 >= 4.0.4)

Call a user function given with an array of parameters

mixed **call_user_func_array** (string *function_name* [, array *paramarr*]) \linebreak

Call a user defined function given by *function_name*, with the parameters in *paramarr*. For example:

```
function debug($var, $val)
    echo "***DEBUGGING\nVARIABLE: $var\nVALUE:";
    if (is_array($val) || is_object($val) || is_resource($val))
        print_r($val);
    else
        echo "\n$val\n";
    echo "***\n";
}

$c = mysql_connect();
$host = $HTTP_SERVER_VARS["SERVER_NAME"];

call_user_func_array ('debug', array("host", $host));
call_user_func_array ('debug', array("c", $c));
call_user_func_array ('debug', array("HTTP_POST_VARS", $HTTP_POST_VARS));
```

See also: `call_user_func()`, `call_user_method()`, `call_user_method_array()`.

Note: This function was added to the CVS code after release of PHP 4.0.4pl1

create_function (PHP 4)

Create an anonymous (lambda-style) function

string **create_function** (string args, string code) \linebreak

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,$a', 'if (strncmp($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 brillling 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 brillling and the slithy toves"
** Look the same to me! (looking at the first 3 chars)
CRCs: -725381282 , 1908338681
similar(a,b) = 11(45.833333333333%)

```

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 >= 4.0.0)

Return an item from the argument list

mixed **func_get_arg** (int *arg_num*) \linebreak

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 >= 4.0.0)

Returns an array comprising a function's argument list

array **func_get_args** (void) \linebreak

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 >= 4.0.0)

Returns the number of arguments passed to the function

int **func_num_args** (void) \linebreak

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 user-defined function.

```
<?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.

function_exists (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Return TRUE if the given function has been defined

bool **function_exists** (string function_name) \linebreak

Checks the list of defined functions, both built-in (internal) and user-defined, for *function_name*. Returns TRUE on success, FALSE on failure.

```
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 (with the image functions being an example). Also note that **function_exists()** will return `FALSE` for constructs, such as `include_once()` and `echo()`.

See also `method_exists()` and `get_defined_functions()`.

get_defined_functions (PHP 4 >= 4.0.4)

Returns an array of all defined functions

array **get_defined_functions** (void) \linebreak

This function returns an multidimensional array containing a list of all defined functions, both built-in (internal) and user-defined. The internal functions will be accessible via `$arr["internal"]`, and the user defined ones using `$arr["user"]` (see example below).

```
function myrow($id, $data) {
    return "<tr><th>$id</th><td>$data</td></tr>\n";
}

$arr = get_defined_functions();

print_r($arr);
```

Will output something along the lines of:

```
Array
(
    [internal] => Array
        (
            [0] => zend_version
            [1] => func_num_args
            [2] => func_get_arg
            [3] => func_get_args
            [4] => strlen
            [5] => strcmp
            [6] => strncmp
            ...
            [750] => bcscale
            [751] => bccomp
        )
)
```

```

[user] => Array
(
    [0] => myrow
)
)

```

See also `get_defined_vars()` and `get_defined_constants()`.

register_shutdown_function (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Register a function for execution on shutdown

```
int register_shutdown_function ( string func ) \linebreak
```

Registers the function named by *func* to be executed when script processing is complete.

Multiple calls to **register_shutdown_function()** can be made, and each will be called in the same order as they were registered. If you call `exit()` within one registered shutdown function, processing will stop completely and no other registered shutdown functions will be called.

The registered shutdown functions are called after the request has been completed (including sending any output buffers), so it is not possible to send output to the browser using `echo()` or `print()`, or retrieve the contents of any output buffers using `ob_get_contents()`.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

register_tick_function (PHP 4 >= 4.0.3)

Register a function for execution on each tick

```
void register_tick_function ( string func [, mixed arg] ) \linebreak
```

Registers the function named by *func* to be executed when a tick is called.

unregister_tick_function (PHP 4 >= 4.0.3)

De-register a function for execution on each tick

void **unregister_tick_function** (string func [, mixed arg]) \linebreak

De-registers the function named by *func* so it is no longer executed when a tick is called.

XXXV. Gettext

The gettext functions implement a NLS (Native Language Support) API which can be used to internationalize your PHP applications. Please see the gettext documentation from your system for a thorough explanation of these functions.

bind_textdomain_codeset (PHP 4 >= 4.2.0)

Specify the character encoding in which the messages from the DOMAIN message catalog will be returned

string **bind_textdomain_codeset** (string domain, string codeset) \linebreak

Warning

This function is currently not documented; only the argument list is available.

bindtextdomain (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Sets the path for a domain

string **bindtextdomain** (string domain, string directory) \linebreak

The **bindtextdomain()** function sets the path for a domain.

dcgettext (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Overrides the domain for a single lookup

string **dcgettext** (string domain, string message, int category) \linebreak

This function allows you to override the current domain for a single message lookup. It also allows you to specify a *category*.

dcngettext (PHP 4 >= 4.2.0)

Plural version of dcgettext

string **dcngettext** (string domain, string msgid1, string msgid2, int n, int category) \linebreak

Warning

This function is currently not documented; only the argument list is available.

gettext (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Override the current domain

```
string gettext ( string domain, string message) \linebreak
```

The **gettext()** function allows you to override the current domain for a single message lookup.

gettext (PHP 4 >= 4.2.0)

Plural version of gettext

```
string gettext ( string domain, string msgid1, string msgid2, int n) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

gettext (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Lookup a message in the current domain

```
string gettext ( string message) \linebreak
```

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"));
?>
```

ngettext (PHP 4 >= 4.2.0)

Plural version of `gettext`

string **ngettext** (string msgid1, string msgid2, int n) \linebreak

Warning

This function is currently not documented; only the argument list is available.

textdomain (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Sets the default domain

string **textdomain** (string text_domain) \linebreak

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 `NULL` as parameter to get the current setting without changing it.

XXXVI. 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.

Warning

If you want to explicitly specify a large integer, specify it as a string. If you don't do that, PHP will interpret the integer-literal first, possibly resulting in loss of precision, even before GMP comes into play.

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_abs (PHP 4 >= 4.0.4)

Absolute value

resource **gmp_abs** (resource a) \linebreak

Returns absolute value of *a*.

gmp_add (PHP 4 >= 4.0.4)

Add numbers

resource **gmp_add** (resource a, resource b) \linebreak

Add two GMP numbers. The result will be a GMP number representing the sum of the arguments.

gmp_and (PHP 4 >= 4.0.4)

Logical AND

resource **gmp_and** (resource a, resource b) \linebreak

Calculates logical AND of two GMP numbers.

gmp_clrbit (PHP 4 >= 4.0.4)

Clear bit

resource **gmp_clrbit** (resource &a, int index) \linebreak

Clears (sets to 0) bit *index* in *a*.

gmp_cmp (PHP 4 >= 4.0.4)

Compare numbers

int **gmp_cmp** (resource a, resource b) \linebreak

Returns a positive value if $a > b$, zero if $a = b$ and negative value if $a < b$.

gmp_com (PHP 4 >= 4.0.4)

Calculates one's complement of a

resource **gmp_com** (resource a) \linebreak

Warning

This function is currently not documented; only the argument list is available.

gmp_div (PHP 4 >= 4.0.4)

Divide numbers

resource **gmp_div** (resource a, resource b) \linebreak

This function is an alias to `gmp_div_q()`.

gmp_div_q (PHP 4 >= 4.0.4)

Divide numbers

resource **gmp_div_q** (resource a, resource b [, int round]) \linebreak

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_qr (PHP 4 >= 4.0.4)

Divide numbers and get quotient and remainder

array **gmp_div_qr** (resource n, resource d [, int round]) \linebreak

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_r (PHP 4 >= 4.0.4)

Remainder of the division of numbers

resource **gmp_div_r** (resource n , resource d [, int $round$]) \linebreak

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_divexact (PHP 4 >= 4.0.4)

Exact division of numbers

resource **gmp_divexact** (resource n , resource d) \linebreak

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_fact (PHP 4 >= 4.0.4)

Factorial

resource **gmp_fact** (int a) \linebreak

Calculates factorial ($a!$) of a .

gmp_gcd (PHP 4 >= 4.0.4)

Calculate GCD

resource **gmp_gcd** (resource a , resource b) \linebreak

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 (PHP 4 >= 4.0.4)

Calculate GCD and multipliers

array **gmp_gcdext** (resource a , resource b) \linebreak

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_hamdist (PHP 4 >= 4.0.4)

Hamming distance

int **gmp_hamdist** (resource a , resource b) \linebreak

Returns the hamming distance between a and b . Both operands should be non-negative.

gmp_init (PHP 4 >= 4.0.4)

Create GMP number

resource **gmp_init** (mixed number) \linebreak

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 (PHP 4 >= 4.0.4)

Convert GMP number to integer

int **gmp_intval** (resource gmpnumber) \linebreak

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_invert (PHP 4 >= 4.0.4)

Inverse by modulo

resource **gmp_invert** (resource a, resource b) \linebreak

Computes the inverse of *a* modulo *b*. Returns `FALSE` if an inverse does not exist.

gmp_jacobi (PHP 4 >= 4.0.4)

Jacobi symbol

int **gmp_jacobi** (resource *a*, resource *p*) \linebreak

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_legendre (PHP 4 >= 4.0.4)

Legendre symbol

int **gmp_legendre** (resource *a*, resource *p*) \linebreak

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_mod (PHP 4 >= 4.0.4)

Modulo operation

resource **gmp_mod** (resource *n*, resource *d*) \linebreak

Calculates *n* modulo *d*. The result is always non-negative, the sign of *d* is ignored.

gmp_mul (PHP 4 >= 4.0.4)

Multiply numbers

resource **gmp_mul** (resource *a*, resource *b*) \linebreak

Multiplies *a* by *b* and returns the result.

gmp_neg (PHP 4 >= 4.0.4)

Negate number

resource **gmp_neg** (resource *a*) \linebreak

Returns $-a$.

gmp_or (PHP 4 >= 4.0.4)

Logical OR

resource **gmp_or** (resource a, resource b) \linebreak

Calculates logical inclusive OR of two GMP numbers.

gmp_perfect_square (PHP 4 >= 4.0.4)

Perfect square check

bool **gmp_perfect_square** (resource a) \linebreak

Returns TRUE if *a* is a perfect square, FALSE otherwise.

See also: `gmp_sqrt()`, `gmp_sqrtrm()`.

gmp_popcount (PHP 4 >= 4.0.4)

Population count

int **gmp_popcount** (resource a) \linebreak

Return the population count of *a*.

gmp_pow (PHP 4 >= 4.0.4)

Raise number into power

resource **gmp_pow** (resource base, int exp) \linebreak

Raise *base* into power *exp*. The case of 0^0 yields 1. *exp* cannot be negative.

gmp_powm (PHP 4 >= 4.0.4)

Raise number into power with modulo

resource **gmp_powm** (resource base, resource exp, resource mod) \linebreak

Calculate (*base* raised into power *exp*) modulo *mod*. If *exp* is negative, result is undefined.

gmp_prob_prime (PHP 4 >= 4.0.4)

Check if number is "probably prime"

int **gmp_prob_prime** (resource *a* [, int *reps*]) \linebreak

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_random (PHP 4 >= 4.0.4)

Random number

resource **gmp_random** (int *limiter*) \linebreak

Generate a random number. The number will be between *limiter* and zero in value. If *limiter* is negative, negative numbers are generated.

gmp_scan0 (PHP 4 >= 4.0.4)

Scan for 0

int **gmp_scan0** (resource *a*, int *start*) \linebreak

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 (PHP 4 >= 4.0.4)

Scan for 1

int **gmp_scan1** (resource *a*, int *start*) \linebreak

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_setbit (PHP 4 >= 4.0.4)

Set bit

resource **gmp_setbit** (resource &a, int index [, bool set_clear]) \linebreak

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_sign (PHP 4 >= 4.0.4)

Sign of number

int **gmp_sign** (resource a) \linebreak

Return sign of *a* - 1 if *a* is positive and -1 if it's negative.

gmp_sqrt (PHP 4 >= 4.0.4)

Square root

resource **gmp_sqrt** (resource a) \linebreak

Calculates square root of *a*.

gmp_sqrtrm (unknown)

Square root with remainder

array **gmp_sqrtrm** (resource a) \linebreak

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_strval (PHP 4 >= 4.0.4)

Convert GMP number to string

string **gmp_strval** (resource gmpnumber [, int base]) \linebreak

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_sub (PHP 4 >= 4.0.4)

Subtract numbers

resource **gmp_sub** (resource a, resource b) \linebreak

Subtracts *b* from *a* and returns the result.

gmp_xor (PHP 4 >= 4.0.4)

Logical XOR

resource **gmp_xor** (resource a, resource b) \linebreak

Calculates logical exclusive OR (XOR) of two GMP numbers.

XXXVII. 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 >= 4.0.0)

Send a raw HTTP header

int **header** (string string [, bool replace]) \linebreak

header() is used to send raw HTTP headers. See the HTTP/1.1 specification (<http://www.w3.org/Protocols/rfc2616/rfc2616>) for more information on HTTP headers.

The optional *replace* parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in `FALSE` as the second argument you can force multiple headers of the same type. For example:

```
header( 'WWW-Authenticate: Negotiate' );
header( 'WWW-Authenticate: NTLM', false );
```

There are two special-case header calls. The first is a header that starts with the string "HTTP/" (case is not significant), which will be used to figure out the HTTP status code to send. For example, if you have configured Apache to use a PHP script to handle requests for missing files (using the `ErrorDocument` directive), you may want to make sure that your script generates the proper status code.

```
<?php
  header("HTTP/1.0 404 Not Found");
?>
```

Note: The HTTP status header line will always be the first sent to the client, regardless of the actual **header()** call being the first or not. The status may be overridden by calling **header()** with a new status line at any time unless the HTTP headers have already been sent.

Note: In PHP 3, this only works when PHP is compiled as an Apache module. You can achieve the same effect using the `Status` header.

```
header("Status: 404 Not Found");
```

The second special case is the "Location:" header. Not only does it send this header back to the browser, but it also returns a REDIRECT (302) status code to the browser unless some 3xx status code has already been set.

```
header("Location: http://www.example.com/"); /* Redirect browser */
exit;                                     /* Make sure that code below does
                                         not get executed when we redirect. */
```

Note: HTTP/1.1 requires an absolute URI as argument to Location: (<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.30>) including the scheme, hostname and absolute path, but some clients accept relative URIs. You can usually use `$_SERVER['HTTP_HOST']`, `$_SERVER['PHP_SELF']` and `dirname()` to make an absolute URI from a relative one yourself:

```
header("Location: http://" . $_SERVER['HTTP_HOST']
      . dirname($_SERVER['PHP_SELF'])
      . "/" . $relative_url);
```

PHP scripts often generate dynamic content 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-store, no-cache, must-revalidate"); // HTTP/1.1
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache"); // HTTP/1.0
```

Note: You may find that your pages aren't cached even if you don't output all of the headers above. There are a number of options that users may be able to set for their browser that change its default caching behavior. By sending the headers above, you should override any settings that may otherwise cause the output of your script to be cached.

Additionally, `session_cache_limiter()` and the `session.cache_limiter` configuration setting can be used to automatically generate the correct caching-related headers when sessions are being used.

Remember that **header()** must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with `include()`, or `require()`, functions, or another file access function, and have spaces or empty lines that are output before **header()** is called. The same problem exists when using a single PHP/HTML file.

```
<?php header ("Content-type: audio/x-pn-realaudio"); ?>
// Broken, note the blank line above
```

Note: In PHP 4, you can use output buffering to get around this problem, with the overhead of all of your output to the browser being buffered in the server until you send it. You can do this by calling `ob_start()` and `ob_end_flush()` in your script, or setting the `output_buffering` configuration directive on in your `php.ini` or server configuration files.

If you want the user to be prompted to save the data you are sending, such as a generated PDF file, you can use the `Content-Disposition` (<http://www.ietf.org/rfc/rfc2183.txt>) header to supply a recommended filename and force the browser to display the save dialog.

```
<?php
header("Content-type: application/pdf");
header("Content-Disposition: attachment; filename=downloaded.pdf");

/* ... output pdf file ... */
```

Note: There is a bug in Microsoft Internet Explorer 4.01 that prevents this from working. There is no workaround. There is also a bug in Microsoft Internet Explorer 5.5 that interferes with this, which can be resolved by upgrading to Service Pack 2 or later.

See also `headers_sent()`, `setcookie()`, and the section on HTTP authentication.

headers_sent (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Returns `TRUE` if headers have been sent

bool **headers_sent** (void) \linebreak

This function returns `TRUE` if the HTTP headers have already been sent, `FALSE` otherwise.

See also `header()`

setcookie (PHP 3, PHP 4 >= 4.0.0)

Send a cookie

int **setcookie** (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]) \linebreak

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", $value);
setcookie ("TestCookie", $value,time()+3600); /* expire in 1 hour */
setcookie ("TestCookie", $value,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. Examples follow how to delete cookies sent in previous example:

Example 2. setcookie() delete examples

```
// set the expiration date to one hour ago
setcookie ("TestCookie", "", time() - 3600);
setcookie ("TestCookie", "", time() - 3600, "~/rasmus/", ".utoronto.ca", 1);
```

Note that the value portion of the cookie will automatically be urlencoded when you send the cookie, and when it is received, it is automatically decoded and assigned to a variable by the same name as the cookie name. To see the contents of our test cookie in a script, simply use one of the following examples:

```
echo $TestCookie;
echo $HTTP_COOKIE_VARS["TestCookie"];
```

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.

XXXVIII. Hyperwave functions

Introduction

Hyperwave has been developed at IICM (<http://www.iicm.edu/>) in Graz. It started with the name Hyper-G and changed to Hyperwave when it was commercialised (If I remember properly it was in 1996).

Hyperwave is not free software. The current version, 4.1, is available at www.hyperwave.com (<http://www.hyperwave.com/>). A time limited version can be ordered for free (30 days).

Hyperwave is an information system similar to a database (HIS, Hyperwave Information Server). Its focus is the storage and management of documents. A document can be any possible piece of data that may as well be stored in file. Each document is accompanied by its object record. The object record contains meta data for the document. The meta data is a list of attributes which can be extended by the user. Certain attributes are always set by the Hyperwave server, other may be modified by the user. An attribute is a name/value pair of the form name=value. The complete object record contains as many of those pairs as the user likes. The name of an attribute does not have to be unique, e.g. a title may appear several times within an object record. This makes sense if you want to specify a title in several languages. In such a case there is a convention, that each title value is preceded by the two letter language abbreviation followed by a colon, e.g. 'en:Title in English' or 'ge:Titel in deutsch'. Other attributes like a description or keywords are potential candidates. You may also replace the language abbreviation by any other string as long as it separated by colon from the rest of the attribute value.

Each object record has native a string representation with each name/value pair separated by a newline. The Hyperwave extension also knows a second representation which is an associated array with the attribute name being the key. Multilingual attribute values itself form another associated array with the key being the language abbreviation. Actually any multiple attribute forms an associated array with the string left to the colon in the attribute value being the key. (This is not fully implemented. Only the attributes Title, Description and Keyword are treated properly yet.)

Besides the documents, all hyper links contained in a document are stored as object records as well. Hyper links which are in a document will be removed from it and stored as individual objects, when the document is inserted into the database. The object record of the link contains information about where it starts and where it ends. In order to gain the original document you will have to retrieve the plain document without the links and the list of links and reinsert them (The functions `hw_pipedocument()` and `hw_gettext()` do this for you. The advantage of separating links from the document is obvious. Once a document to which a link is pointing to changes its name, the link can easily be modified accordingly. The document containing the link is not affected at all. You may even add a link to a document without modifying the document itself.

Saying that `hw_pipedocument()` and `hw_gettext()` do the link insertion automatically is not as simple as it sounds. Inserting links implies a certain hierarchy of the documents. On a web server this is given by the file system, but Hyperwave has its own hierarchy and names do not reflect the position of an object in that hierarchy. Therefore creation of links first of all requires a mapping from the Hyperwave hierarchy and namespace into a web hierarchy respective web namespace. The fundamental difference between Hyperwave and the web is the clear distinction between names and hierarchy in Hyperwave. The name does not contain any information about the objects position in the hierarchy. In the web the name also contains the information on where the object is located in the hierarchy. This leads to two possible ways

of mapping. Either the Hyperwave hierarchy and name of the Hyperwave object is reflected in the URL or the name only. To make things simple the second approach is used. Hyperwave object with name 'my_object' is mapped to 'http://host/my_object' disregarding where it resides in the Hyperwave hierarchy. An object with name 'parent/my_object' could be the child of 'my_object' in the Hyperwave hierarchy, though in a web namespace it appears to be just the opposite and the user might get confused. This can only be prevented by selecting reasonable object names.

Having made this decision a second problem arises. How do you involve PHP? The URL `http://host/my_object` will not call any PHP script unless you tell your web server to rewrite it to e.g. `'http://host/php3_script/my_object'` and the script 'php3_script' evaluates the `$PATH_INFO` variable and retrieves the object with name 'my_object' from the Hyperwave server. There is just one little drawback which can be fixed easily. Rewriting any URL would not allow any access to other document on the web server. A PHP script for searching in the Hyperwave server would be impossible. Therefore you will need at least a second rewriting rule to exclude certain URLs like all e.g. starting with `http://host/Hyperwave`. This is basically sharing of a namespace by the web and Hyperwave server.

Based on the above mechanism links are inserted into documents.

It gets more complicated if PHP is not run as a server module or CGI script but as a standalone application e.g. to dump the content of the Hyperwave server on a CD-ROM. In such a case it makes sense to retain the Hyperwave hierarchy and map in 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: An unique integer value for each object in the Hyperwave server. It is also one of the attributes of the object record (ObjectID). Object ids are often used as an input parameter to specify an object.
- object record: A string with attribute-value pairs of the form attribute=value. The pairs are separated by a carriage return from each other. An object record can easily be converted into an object array with `hw_object2array()`. Several functions return object records. The names of those functions end with `obj`.
- object array: An associated array with all attributes of an object. The key is the attribute name. If an

attribute occurs more than once in an object record it will result in another indexed or associated array. Attributes which are language depended (like the title, keyword, description) will form an associated array with the key set to the language abbreviation. All other multiple attributes will form an indexed array. PHP functions never return object arrays.

- `hw_document`: This is a complete new data type which holds the actual document, e.g. HTML, PDF etc. It is somewhat optimised for HTML documents but may be used for any format.

Several functions which return an array of object records do also return an associated array with statistical information about them. The array is the last element of the object record array. The statistical array contains the following entries:

Hidden

Number of object records with attribute `PresentationHints` set to `Hidden`.

CollectionHead

Number of object records with attribute `PresentationHints` set to `CollectionHead`.

FullCollectionHead

Number of object records with attribute `PresentationHints` set to `FullCollectionHead`.

CollectionHeadNr

Index in array of object records with attribute `PresentationHints` set to `CollectionHead`.

FullCollectionHeadNr

Index in array of object records with attribute `PresentationHints` set to `FullCollectionHead`.

Total

Total: Number of object records.

Integration with Apache

The Hyperwave extension is best used when PHP is compiled as an Apache module. In such a case the underlying Hyperwave server can be hidden from users almost completely if Apache uses its rewriting engine. The following instructions will explain this.

Since PHP with Hyperwave support built into Apache is intended to replace the native Hyperwave solution based on Wavemaster I will assume that the Apache server will only serve as a Hyperwave web interface. This is not necessary but it simplifies the configuration. The concept is quite simple. First of all you need a PHP script which evaluates the `PATH_INFO` variable and treats its value as the name of a Hyperwave object. Let's call this script `'Hyperwave'`. The URL `http://your.hostname/Hyperwave/name_of_object` would then return the Hyperwave object with the name `'name_of_object'`. Depending on the type of the object the script has to react accordingly. If it is a collection, it will probably return a list of children. If it is a document it will return the mime type and the

content. A slight improvement can be achieved if the Apache rewriting engine is used. From the users point of view it would be more straight forward if the URL `http://your.hostname/name_of_object` would return the object. The rewriting rule is quite easy:

```
RewriteRule ^/(.*) /usr/local/apache/htdocs/HyperWave/$1 [L]
```

Now every URL relates to an object in the Hyperwave server. This causes a simple to solve problem. There is no way to execute a different script, e.g. for searching, than the 'Hyperwave' script. This can be fixed with another rewriting rule like the following:

```
RewriteRule ^/hw/(.*) /usr/local/apache/htdocs/hw/$1 [L]
```

This will reserve the directory `/usr/local/apache/htdocs/hw` for additional scripts and other files. Just make sure this rule is evaluated before the one above. There is just a little drawback: all Hyperwave objects whose name starts with 'hw/' will be shadowed. So, make sure you don't use such names. If you need more directories, e.g. for images just add more rules or place them all in one directory. Finally, don't forget to turn on the rewriting engine with

```
RewriteEngine on
```

My experiences have shown that you will need the following scripts:

- to return the object itself
- to allow searching
- to identify yourself
- to set your profile
- one for each additional function like to show the object attributes, to show information about users, to

show the status of the server, etc.

Todo

There are still some things todo:

- The `hw_InsertDocument` has to be split into `hw_insertobject()` and **`hw_putdocument()`**.
- The names of several functions are not fixed, yet.
- Most functions require the current connection as its first parameter. This leads to a lot of typing, which is quite often not necessary if there is just one open connection. A default connection will improve this.
- Conversion from object record into object array needs to handle any multiple attribute.

hw_Array2Objrec (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

convert attributes from object array to object record

```
string hw_array2objrec ( array object_array ) \linebreak
```

Converts an *object_array* into an object record. Multiple attributes like 'Title' in different languages are treated properly.

See also `hw_objrec2array()`.

hw_changeobject (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Changes attributes of an object (obsolete)

```
void hw_changeobject ( int link, int objid, array attributes ) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

hw_Children (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object ids of children

```
array hw_children ( int connection, int objectID ) \linebreak
```

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 >= 4.0.0)

object records of children

```
array hw_childrenobj ( int connection, int objectID ) \linebreak
```

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 >= 4.0.0)

closes the Hyperwave connection

int **hw_close** (int connection) \linebreak

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 >= 4.0.0)

opens a connection

int **hw_connect** (string host, int port, string username, string password) \linebreak

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_connection_info (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Prints information about the connection to Hyperwave server

void **hw_connection_info** (int link) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_Cp (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

copies objects

int **hw_cp** (int connection, array object_id_array, int destination id) \linebreak

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 >= 4.0.0)

deletes object

`int hw_deleteobject (int connection, int object_to_delete) \linebreak`

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 >= 4.0.0)

object id object belonging to anchor

`int hw_docbyanchor (int connection, int anchorID) \linebreak`

Returns an th object id of the document to which *anchorID* belongs.

hw_DocByAnchorObj (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object record object belonging to anchor

`string hw_docbyanchorobj (int connection, int anchorID) \linebreak`

Returns an th object record of the document to which *anchorID* belongs.

hw_Document_Attributes (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object record of hw_document

`string hw_document_attributes (int hw_document) \linebreak`

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 >= 4.0.0)

body tag of `hw_document`

string **hw_document_bodytag** (int `hw_document`) \linebreak

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 >= 4.0.0)

returns content of `hw_document`

string **hw_document_content** (int `hw_document`) \linebreak

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_document_setcontent()`.

hw_Document_SetContent (PHP 4 >= 4.0.0)

sets/replaces content of `hw_document`

string **hw_document_setcontent** (int `hw_document`, string `content`) \linebreak

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 >= 4.0.0)

size of `hw_document`

int **hw_document_size** (int hw_document) \linebreak

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_dummy (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Hyperwave dummy function

string **hw_dummy** (int link, int id, int msgid) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_EditText (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

retrieve text document

int **hw_edittest** (int connection, int hw_document) \linebreak

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_free_document()`, `hw_document_bodytag()`, `hw_document_size()`, `hw_output_document()`, `hw_gettext()`.

hw_Error (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

error number

int **hw_error** (int connection) \linebreak

Returns the last error number. If the return value is 0 no error has occurred. The error relates to the last command.

hw_ErrorMsg (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

returns error message

string **hw_errormsg** (int connection) \linebreak

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_Free_Document (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

frees hw_document

int **hw_free_document** (int hw_document) \linebreak

Frees the memory occupied by the Hyperwave document.

hw_GetAnchors (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object ids of anchors of document

array **hw_getanchors** (int connection, int objectID) \linebreak

Returns an array of object ids with anchors of the document with object ID *objectID*.

hw_GetAnchorsObj (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object records of anchors of document

array **hw_getanchorsobj** (int connection, int objectID) \linebreak

Returns an array of object records with anchors of the document with object ID *objectID*.

hw_GetAndLock (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

return bject record and lock object

string **hw_getandlock** (int connection, int objectID) \linebreak

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_GetChildColl (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object ids of child collections

array **hw_getchildcoll** (int connection, int objectID) \linebreak

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_children()`, and `hw_getchilddoccoll()`.

hw_GetChildCollObj (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object records of child collections

array **hw_getchildcollobj** (int connection, int objectID) \linebreak

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_GetChildDocColl (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object ids of child documents of collection

array **hw_getchilddoccoll** (int connection, int objectID) \linebreak

Returns array of object ids for child documents of a collection.

See also `hw_children()`, and `hw_getchildcoll()`.

hw_GetChildDocCollObj (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object records of child documents of collection

array **hw_getchilddoccollobj** (int connection, int objectID) \linebreak

Returns an array of object records for child documents of a collection.

See also `hw_childrenobj()`, and `hw_getchildcollobj()`.

hw_GetObject (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object record

array **hw_getobject** (int connection, [int|array] objectID, string query) \linebreak

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_GetObjectByQuery (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

search object

array **hw_getobjectbyquery** (int connection, string query, int max_hits) \linebreak

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_GetObjectByQueryColl (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

search object in collection

array **hw_getobjectbyquerycoll** (int connection, int objectID, string query, int max_hits) \linebreak

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 >= 4.0.0)

search object in collection

array **hw_getobjectbyquerycollobj** (int connection, int objectID, string query, int max_hits) \linebreak

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_GetObjectByQueryObj (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

search object

array **hw_getobjectbyqueryobj** (int connection, string query, int max_hits) \linebreak

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_GetParents (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

object ids of parents

array **hw_getparents** (int connection, int objectID) \linebreak

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 >= 4.0.0)

object records of parents

array **hw_getparentsobj** (int connection, int objectID) \linebreak

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_getrellink (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Get link from source to dest relative to rootid

string **hw_getrellink** (int link, int rootid, int sourceid, int destid) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_GetRemote (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Gets a remote document

int **hw_getremote** (int connection, int objectID) \linebreak

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 >= 4.0.0)

Gets children of remote document

int **hw_getremotechildren** (int connection, string object record) \linebreak

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 >= 4.0.0)

Returns anchors pointing at object

array **hw_getsrcbydestobj** (int connection, int objectID) \linebreak

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_GetText (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

retrieve text document

int **hw_gettext** (int connection, int objectID [, mixed rootID/prefix]) \linebreak

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: ``. 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_free_document()`, `hw_document_bodytag()`, `hw_document_size()`, and `hw_output_document()`.

hw_getusername (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

name of currently logged in user

string **hw_getusername** (int connection) \linebreak

Returns the username of the connection.

hw_Identify (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

identifies as user

int **hw_identify** (string username, string password) \linebreak

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 >= 4.0.0)

check if object ids in collections

array **hw_incollections** (int connection, array object_id_array, array collection_id_array, int return_collections) \linebreak

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 >= 4.0.0)

info about connection

string **hw_info** (int connection) \linebreak

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 >= 4.0.0)

insert collection

int **hw_inscoll** (int connection, int objectID, array object_array) \linebreak

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 >= 4.0.0)

insert document

int **hw_insdoc** (int connection, int parentID, string object_record, string text) \linebreak

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_insertanchors (PHP 4 >= 4.0.4)

Inserts only anchors into text

string **hw_insertanchors** (int hwdoc, array anchorecs, array dest [, array urlprefixes]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_InsertDocument (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

upload any document

int **hw_insertdocument** (int connection, int parent_id, int hw_document) \linebreak

Uploads a document into the collection with *parent_id*. The document has to be created before with `hw_new_document()`. 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 >= 4.0.0)

inserts an object record

int **hw_insertobject** (int connection, string object rec, string parameter) \linebreak

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 >= 4.0.0)

Maps global id on virtual local id

int **hw_mapid** (int connection, int server id, int object id) \linebreak

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 >= 4.0.0)

modifies object record

```
int hw_modifyobject ( int connection, int object_to_change, array remove, array add, int mode) \linebreak
```

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 completely removed. You will get an error message 'Change of base attribute' (not clear when this happens). Therefore you will always have to add a new Name first and then remove the old one.

Note: You may not surround this function by calls to `hw_getandlock()` and `hw_unlock()`. **hw_modifyobject()** does this internally.

Returns `TRUE` if no error occurs otherwise `FALSE`.

hw_Mv (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

moves objects

int **hw_mv** (int connection, array object id array, int source id, int destination id) \linebreak

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_New_Document (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

create new document

int hw_new_document (string object_record, string document_data, int document_size) \linebreak

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_free_document()`, `hw_document_size()`, `hw_document_bodytag()`, `hw_output_document()`, and `hw_insertdocument()`.

hw_Objrec2Array (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

convert attributes from object record to object array

array hw_objrec2array (string object_record [, array format]) \linebreak

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 >= 4.0.0)

prints hw_document

int hw_output_document (int hw_document) \linebreak

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 >= 4.0.0)

make a persistent database connection

int hw_pconnect (string host, int port, string username, string password) \linebreak

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 >= 4.0.0)

retrieve any document

int **hw_pipedocument** (int connection, int objectID) \linebreak

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_free_document()`, `hw_document_size()`, `hw_document_bodytag()`, and `hw_output_document()`.

hw_Root (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

root object id

int **hw_root** () \linebreak

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_setlinkroot (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Set the id to which links are calculated

void **hw_setlinkroot** (int link, int rootid) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_stat (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Returns status string

string **hw_stat** (int link) \linebreak

Warning

This function is currently not documented; only the argument list is available.

hw_Unlock (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

unlock object

int **hw_unlock** (int connection, int objectID) \linebreak

Unlocks a document, so other users regain access.

See also `hw_getandlock()`.

hw_Who (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

List of currently logged in users

int **hw_who** (int connection) \linebreak

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.

XXXIX. Hyperwave API functions

Introduction

Hyperwave has been developed at IICM (<http://www.iicm.edu/>) in Graz. It started with the name Hyper-G and changed to Hyperwave when it was commercialised (If I remember properly it was in 1996).

Hyperwave is not free software. The current version, 5.5, 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.

Requirements

Since 2001 there is a Hyperwave SDK available. It supports Java, JavaScript and C++. This PHP Extension is based on the C++ interface. In order to activate the hwapi support in PHP you will have to install the Hyperwave SDK first and configure PHP with `--with-hwapi=<dir>`.

Classes

The API provided by the HW_API extension is fully object oriented. It is very similar to the C++ interface of the Hyperwave SDK. It consist of the following classes.

- HW_API
- HW_API_Object
- HW_API_Attribute
- HW_API_Error
- HW_API_Content
- HW_API_Reason

Some basic classes like HW_API_String, HW_API_String_Array, etc., which exist in the Hyperwave SDK have not been implemented since PHP has powerful replacements for them.

Each class has certain method, whose names are identical to its counterparts in the Hyperwave SDK. Passing arguments to this function differs from all the other PHP Extension but is close to the C++ API of the HW SDK. Instead of passing several parameters they are all put into an associated array and passed

as one paramter. The names of the keys are identical to those documented in the HW SDK. The common parameters are listed below. If other parameters are required they will be documented if needed.

- `objectIdentifier` The name or id of an object, e.g. "rootcollection", "0x873A8768 0x00000002".
- `parentIdentifier` The name or id of an object which is considered to be a parent.
- `object` An instance of class `HW_API_Object`.
- `parameters` An instance of class `HW_API_Object`.
- `version` The version of an object.
- `mode` An integer value determine the way an operation is executed.
- `attributeSelector` Any array of strings, each containing a name of an attribute. This is used if you retrieve the object record and want to include certain attributes.
- `objectQuery` A query to select certain object out of a list of objects. This is used to reduce the number of objects which was delivered by a function like `hw_api->children()` or `hw_api->find()`.

Integration with Apache

The integration with Apache and possible other servers is already described in the Hyperwave Modul which has been the first extension to connect a Hyperwave Server.

hw_api_attribute (unknown)

Creates instance of class hw_api_attribute

object **attribute** ([string name [, string value]]) \linebreak

Creates a new instance of hw_api_attribute with the given name and value.

hw_api_attribute->key (unknown)

Returns key of the attribute

string **key** (void) \linebreak

Returns the name of the attribute.

See also hwapi_attribute_value().

hw_api_attribute->langdepvalue (unknown)

Returns value for a given language

string **langdepvalue** (string language) \linebreak

Returns the value in the given language of the attribute.

See also hwapi_attribute_value().

hw_api_attribute->value (unknown)

Returns value of the attribute

string **value** (void) \linebreak

Returns the value of the attribute.

See also hwapi_attribute_key(), hwapi_attribute_values().

hw_api_attribute->values (unknown)

Returns all values of the attribute

array **values** (void) \linebreak

Returns all values of the attribute as an array of strings.

See also `hwapi_attribute_value()`.

hw_api->checkin (unknown)

Checks in an object

object **checkin** (array parameter) \linebreak

This function checks in an object or a whole hierarchy of objects. The parameters array contains the required element 'objectIdentifier' and the optional element 'version', 'comment', 'mode' and 'objectQuery'. 'version' sets the version of the object. It consists of the major and minor version separated by a period. If the version is not set, the minor version is incremented. 'mode' can be one of the following values:

HW_API_CHECKIN_NORMAL

Checks in and commits the object. The object must be a document.

HW_API_CHECKIN_RECURSIVE

If the object to check in is a collection, all children will be checked in recursively if they are documents. Trying to check in a collection would result in an error.

HW_API_CHECKIN_FORCE_VERSION_CONTROL

Checks in an object even if it is not under version control.

HW_API_CHECKIN_REVERT_IF_NOT_CHANGED

Check if the new version is different from the last version. Unless this is the case the object will be checked in.

HW_API_CHECKIN_KEEP_TIME_MODIFIED

Keeps the time modified from the most recent object.

HW_API_CHECKIN_NO_AUTO_COMMIT

The object is not automatically committed on checkin.

See also `hwapi_checkout()`.

hw_api->checkout (unknown)

Checks out an object

object **checkout** (array parameter) \linebreak

This function checks out an object or a whole hierarchy of objects. The parameters array contains the required element 'objectIdentifier' and the optional element 'version', 'mode' and 'objectQuery'. 'mode' can be one of the following values:

HW_API_CHECKIN_NORMAL

Checks out an object. The object must be a document.

HW_API_CHECKIN_RECURSIVE

If the object to check out is a collection, all children will be checked out recursively if they are documents. Trying to check out a collection would result in an error.

See also `hwapi_checkin()`.

hw_api->children (unknown)

Returns children of an object

array **children** (array parameter) \linebreak

Retrieves the children of a collection or the attributes of a document. The children can be further filtered by specifying an object query. The parameter array contains the required elements 'objectIdentifier' and the optional elements 'attributeSelector' and 'objectQuery'.

The return value is an array of objects of type `HW_API_Object` or `HW_API_Error`.

See also `hwapi_parents()`.

hw_api->content (unknown)

Returns content of an object

object **content** (array parameter) \linebreak

This function returns the content of a document as an object of type `hw_api_content`. The parameter array contains the required elements 'objectIdentifier' and the optional element 'mode'. The mode can be one of the constants `HW_API_CONTENT_ALLLINKS`, `HW_API_CONTENT_REACHABLELINKS` or `HW_API_CONTENT_PLAIN`. `HW_API_CONTENT_ALLLINKS` means to insert all anchors even if the destination is not reachable. `HW_API_CONTENT_REACHABLELINKS` tells `hw_api_content()` to insert only reachable links and `HW_API_CONTENT_PLAIN` will lead to document without any links.

hw_api_content->mimetype (unknown)

Returns mimetype

string **mimetype** (void) \linebreak

Returns the mimetype of the content.

hw_api_content->read (unknown)

Read content

string **read** (string buffer, integer len) \linebreak

Reads *len* bytes from the content into the given buffer.

hw_api->copy (unknown)

Copies physically

object **copy** (array parameter) \linebreak

This function will make a physical copy including the content if it exists and returns the new object or an error object. The parameter array contains the required elements 'objectIdentifier' and 'destinationParentIdentifier'. The optional parameter is 'attributeSelector'

See also hwapi_move(), hwapi_link().

hw_api->dbstat (unknown)

Returns statistics about database server

object **dbstat** (array parameter) \linebreak

See also hwapi_dcstat(), hwapi_hwstat(), hwapi_ftstat().

hw_api->dcstat (unknown)

Returns statistics about document cache server

object **dcstat** (array parameter) \linebreak

See also `hwapi_hwstat()`, `hwapi_dbstat()`, `hwapi_ftstat()`.

hw_api->dstanchors (unknown)

Returns a list of all destination anchors

object **dstanchors** (array parameter) \linebreak

Retrieves all destination anchors of an object. The parameter array contains the required element 'objectIdentifier' and the optional elements 'attributeSelector' and 'objectQuery'.

See also `hwapi_srcanchors()`.

hw_api->dstofsrcanchors (unknown)

Returns destination of a source anchor

object **dstofsrcanchors** (array parameter) \linebreak

Retrieves the destination object pointed by the specified source anchors. The destination object can either be a destination anchor or a whole document. The parameters array contains the required element 'objectIdentifier' and the optional element 'attributeSelector'.

See also `hwapi_srcanchors()`, `hwapi_dstanchors()`, `hwapi_objectbyanchor()`.

hw_api_error->count (unknown)

Returns number of reasons

int **count** (void) \linebreak

Returns the number of error reasons.

See also `hwapi_error_reason()`.

hw_api_error->reason (unknown)

Returns reason of error

object **reason** (void) \linebreak

Returns the first error reason.

See also `hwapi_error_count()`.

hw_api->find (unknown)

Search for objects

array **find** (array parameter) \linebreak

This functions searches for objects either by executing a key or/and full text query. The found objects can further be filtered by an optional object query. They are sorted by their importance. The second search operation is relatively slow and its result can be limited to a certain number of hits. This allows to perform an incremental search, each returning just a subset of all found documents, starting at a given index. The parameter array contains the 'keyquery' or/and 'fulltextquery' depending on who you would like to search. Optional parameters are 'objectquery', 'scope', 'lanugages' and 'attributeselector'. In case of an incremental search the optional parameters 'startIndex', 'numberOfObjectsToGet' and 'exactMatchUnit' can be passed.

hw_api->ftstat (unknown)

Returns statistics about fulltext server

object **ftstat** (array parameter) \linebreak

See also `hwapi_dcstat()`, `hwapi_dbstat()`, `hwapi_hwstat()`.

hwapi_hgcsp (unknown)

Returns object of class `hw_api`

object **hwapi_hgcsp** (string hostname [, int port]) \linebreak

Opens a connection to the Hyperwave server on host *hostname*. The protocol used is HGCSPP. If you do not pass a port number, 418 is used.

See also `hwapi_hwtp()`.

hw_api->hwstat (unknown)

Returns statistics about Hyperwave server

object **hwstat** (array parameter) \linebreak

See also `hwapi_dcstat()`, `hwapi_dbstat()`, `hwapi_ftstat()`.

hw_api->identify (unknown)

Log into Hyperwave Server

object **identify** (array parameter) \linebreak

Logs into the Hyperwave Server. The parameter array must contain the elements 'username' und 'password'.

The return value will be an object of type `HW_API_Error` if identification failed or `TRUE` if it was successful.

hw_api->info (unknown)

Returns information about server configuration

object **info** (array parameter) \linebreak

See also `hwapi_dcstat()`, `hwapi_dbstat()`, `hwapi_ftstat()`, `hwapi_hwstat()`.

hw_api->insert (unknown)

Inserts a new object

object **insert** (array parameter) \linebreak

Insert a new object. The object type can be user, group, document or anchor. Depending on the type other object attributes has to be set. The parameter array contains the required elements 'object' and 'content' (if the object is a document) and the optional parameters 'parameters', 'mode' and 'attributeSelector'.

The 'object' must contain all attributes of the object. 'parameters' is an object as well holding further attributes like the destination (attribute key is 'Parent'). 'content' is the content of the document. 'mode' can be a combination of the following flags:

`HW_API_INSERT_NORMAL`

The object is inserted into the server.

`HW_API_INSERT_FORCE-VERSION-CONTROL`

`HW_API_INSERT_AUTOMATIC-CHECKOUT`

`HW_API_INSERT_PLAIN`

HW_API_INSERT_KEEP_TIME_MODIFIED

HW_API_INSERT_DELAY_INDEXING

See also `hwapi_replace()`.

hw_api->insertanchor (unknown)

Inserts a new object of type anchor

object **insertanchor** (array parameter) \linebreak

This function is a shortcut for `hwapi_insert()`. It inserts an object of type anchor and sets some of the attributes required for an anchor. The parameter array contains the required elements 'object' and 'documentIdentifier' and the optional elements 'destinationIdentifier', 'parameter', 'hint' and 'attributeSelector'. The 'documentIdentifier' specifies the document where the anchor shall be inserted. The target of the anchor is set in 'destinationIdentifier' if it already exists. If the target does not exist the element 'hint' has to be set to the name of object which is supposed to be inserted later. Once it is inserted the anchor target is resolved automatically.

See also `hwapi_insertdocument()`, `hwapi_insertcollection()`, `hwapi_insert()`.

hw_api->insertcollection (unknown)

Inserts a new object of type collection

object **insertcollection** (array parameter) \linebreak

This function is a shortcut for `hwapi_insert()`. It inserts an object of type collection and sets some of the attributes required for a collection. The parameter array contains the required elements 'object' and 'parentIdentifier' and the optional elements 'parameter' and 'attributeSelector'. See `hwapi_insert()` for the meaning of each element.

See also `hwapi_insertdocument()`, `hwapi_insertanchor()`, `hwapi_insert()`.

hw_api->insertdocument (unknown)

Inserts a new object of type document

object **insertdocument** (array parameter) \linebreak

This function is a shortcut for `hwapi_insert()`. It inserts an object with content and sets some of the attributes required for a document. The parameter array contains the required elements 'object', 'parentIdentifier' and 'content' and the optional elements 'mode', 'parameter' and 'attributeSelector'. See `hwapi_insert()` for the meaning of each element.

See also `hwapi_insert()` `hwapi_insertanchor()`, `hwapi_insertcollection()`.

hw_api->link (unknown)

Creates a link to an object

object **link** (array parameter) \linebreak

Creates a link to an object. Accessing this link is like accessing the object to links points to. The parameter array contains the required elements 'objectIdentifier' and 'destinationParentIdentifier'. 'destinationParentIdentifier' is the target collection.

The function returns true on success or an error object.

See also `hwapi_copy()`.

hw_api->lock (unknown)

Locks an object

object **lock** (array parameter) \linebreak

Locks an object for exclusive editing by the user calling this function. The object can be only unlocked by this user or the system user. The parameter array contains the required element 'objectIdentifier' and the optional parameters 'mode' and 'objectquery'. 'mode' determines how an object is locked. `HW_API_LOCK_NORMAL` means, an object is locked until it is unlocked.

`HW_API_LOCK_RECURSIVE` is only valid for collection and locks all objects within the collection and possible subcollections. `HW_API_LOCK_SESSION` means, an object is locked only as long as the session is valid.

See also `hwapi_unlock()`.

hw_api->move (unknown)

Moves object between collections

object **move** (array parameter) \linebreak

See also `hw_objrec2array()`.

hw_api_content (unknown)

Create new instance of class hw_api_content

string **content** (string content, string mimetype) \linebreak

Creates a new content object from the string *content*. The mimetype is set to *mimetype*.

hw_api->object (unknown)

Retrieve attribute information

object **hw_api->object** (array parameter) \linebreak

This function retrieves the attribute information of an object of any version. It will not return the document content. The parameter array contains the required elements 'objectIdentifier' and the optional elements 'attributeSelector' and 'version'.

The returned object is an instance of class HW_API_Object on success or HW_API_Error if an error occurred.

This simple example retrieves an object and checks for errors.

Example 1. Retrieve an object

```

<?php
function handle_error($error) {
    $reason = $error->reason(0);
    echo "Type: <B>";
    switch($reason->type()) {
        case 0:
            echo "Error";
            break;
        case 1:
            echo "Warning";
            break;
        case 2:
            echo "Message";
            break;
    }
    echo "</B><BR>\n";
    echo "Description: ".$reason->description("en")."<BR>\n";
}

function list_attr($obj) {
    echo "<TABLE>\n";
    $count = $obj->count();
    for($i=0; $i<$count; $i++) {
        $attr = $obj->attribute($i);
    }
}

```

```

        printf(" <TR><TD ALIGN=right bgcolor=#c0c0c0><B>%s</B></TD><TD bgcolor=#F0F0F0>%s</TD>>
                $attr->key(), $attr->value());
    }
    echo "</TABLE>\n";
}

$hwapi = hwapi_hgcsp($g_config[HOSTNAME]);
$params = array("objectIdentifier"=>"rootcollection", "attributeSelector"=>array("Title", "Name"));
$root = $hwapi->object($params);
if(get_class($root) == "HW_API_Error") {
    handle_error($root);
    exit;
}
list_attr($root);
?>

```

See also `hwapi_content()`.

hw_api_object->assign (unknown)

Clones object

object **assign** (array parameter) \linebreak

Clones the attributes of an object.

hw_api_object->attreditable (unknown)

Checks whether an attribute is editable

bool **attreditable** (array parameter) \linebreak

hw_api_object->count (unknown)

Returns number of attributes

int **count** (array parameter) \linebreak

hw_api_object->insert (unknown)

Inserts new attribute

bool **insert** (object attribute) \linebreak

Adds an attribute to the object. Returns true on success and otherwise false.

See also hwapi_object_remove().

hw_api_object (unknown)

Creates a new instance of class hw_api_object

object **hw_api_object** (array parameter) \linebreak

See also hwapi_lock().

hw_api_object->remove (unknown)

Removes attribute

bool **remove** (string name) \linebreak

Removes the attribute with the given name. Returns true on success and otherwise false.

See also hwapi_object_insert().

hw_api_object->title (unknown)

Returns the title attribute

string **title** (array parameter) \linebreak

hw_api_object->value (unknown)

Returns value of attribute

string **value** (string name) \linebreak

Returns the value of the attribute with the given name or false if an error occurred.

hw_api->objectbyanchor (unknown)

Returns the object an anchor belongs to

object **objectbyanchor** (array parameter) \linebreak

This function retrieves an object the specified anchor belongs to. The parameter array contains the required element 'objectIdentifier' and the optional element 'attributeSelector'.

See also **hwapi_dstofsrcanchor()**, **hwapi_srcanchors()**, **hwapi_dstanchors()**.

hw_api->parents (unknown)

Returns parents of an object

array **parents** (array parameter) \linebreak

Retrieves the parents of an object. The parents can be further filtered by specifying an object query. The parameter array contains the required elements 'objectIdentifier' and the optional elements 'attributeselector' and 'objectquery'.

The return value is an array of objects of type `HW_API_Object` or `HW_API_Error`.

See also **hwapi_children()**.

hw_api_reason->description (unknown)

Returns description of reason

string **description** (void) \linebreak

Returns the description of a reason

hw_api_reason->type (unknown)

Returns type of reason

object **type** (void) \linebreak

Returns the type of a reason.

hw_api->remove (unknown)

Delete an object

object **remove** (array parameter) \linebreak

This function removes an object from the specified parent. Collections will be removed recursively. You can pass an optional object query to remove only those objects which match the query. An object will be deleted physically if it is the last instance. The parameter array contains the required elements 'objectIdentifier' and 'parentIdentifier'. If you want to remove a user or group 'parentIdentifier' can be skipped. The optional parameter 'mode' determines how the deletion is performed. In normal mode the object will not be removed physically until all instances are removed. In physical mode all instances of the object will be deleted immediately. In removelinks mode all references to and from the objects will be deleted as well. In nonrecursive the deletion is not performed recursive. Removing a collection which is not empty will cause an error.

See also hwapi_move().

hw_api->replace (unknown)

Replaces an object

object **replace** (array parameter) \linebreak

Replaces the attributes and the content of an object The parameter array contains the required elements 'objectIdentifier' and 'object' and the optional parameters 'content', 'parameters', 'mode' and 'attributeSelector'. 'objectIdentifier' contains the object to be replaced. 'object' contains the new object. 'content' contains the new content. 'parameters' contain extra information for HTML documents. HTML_Language is the letter abbreviation of the language of the title. HTML_Base sets the base attribute of the HTML document. 'mode' can be a combination of the following flags:

HW_API_REPLACE_NORMAL

The object on the server is replace with the object passed.

HW_API_REPLACE_FORCE_VERSION_CONTROL

HW_API_REPLACE_AUTOMATIC_CHECKOUT

HW_API_REPLACE_AUTOMATIC_CHECKIN

HW_API_REPLACE_PLAIN

HW_API_REPLACE_REVERT_IF_NOT_CHANGED

HW_API_REPLACE_KEEP_TIME_MODIFIED

See also `hwapi_insert()`.

hw_api->setcommittedversion (unknown)

Commits version other than last version

object **setcommittedversion** (array parameter) \linebreak

Commits a version of a document. The committed version is the one which is visible to users with read access. By default the last version is the committed version.

See also `hwapi_checkin()`, `hwapi_checkout()`, **`hwapi_revert()`**.

hw_api->srcanchors (unknown)

Returns a list of all source anchors

object **srcanchors** (array parameter) \linebreak

Retrieves all source anchors of an object. The parameter array contains the required element 'objectIdentifier' and the optional elements 'attributeSelector' and 'objectQuery'.

See also `hwapi_dstanchors()`.

hw_api->srcsofdst (unknown)

Returns source of a destination object

object **srcsofdst** (array parameter) \linebreak

Retrieves all the source anchors pointing to the specified destination. The destination object can either be a destination anchor or a whole document. The parameters array contains the required element 'objectIdentifier' and the optional element 'attributeSelector' and 'objectQuery'. The function returns an array of objects or an error.

See also **`hwapi_dstofsrcanchor()`**.

hw_api->unlock (unknown)

Unlocks a locked object

object **unlock** (array parameter) \linebreak

Unlocks a locked object. Only the user who has locked the object and the system user may unlock an object. The parameter array contains the required element 'objectIdentifier' and the optional parameters 'mode' and 'objectquery'. The meaning of 'mode' is the same as in function hwapi_lock().

Returns true on success or an object of class HW_API_Error.

See also hwapi_lock().

hw_api->user (unknown)

Returns the own user object

object **user** (array parameter) \linebreak

See also hwapi_userlist().

hw_api->userlist (unknown)

Returns a list of all logged in users

object **userlist** (array parameter) \linebreak

See also hwapi_user().

XL. ICAP Functions [deprecated]

To get these functions to work, you have to compile PHP with `--with-icap`. That requires the icap library to be installed, which is not longer supported and available.

Note: Icap will be removed in near future. Neither this module, nor those versions of icap library are supported any longer. If you want to use calendar capabilities in php, use mcal instead.

icap_close (unknown)

Close an ICAP stream

int **icap_close** (int icap_stream [, int flags]) \linebreak

Closes the given icap stream.

icap_create_calendar (PHP 4 >= 4.0.0)

Create a new calendar

string **icap_create_calendar** (int stream_id, string calendar) \linebreak

Warning

This function is currently not documented; only the argument list is available.

icap_delete_calendar (PHP 4 >= 4.0.0)

Delete a calendar

string **icap_delete_calendar** (int stream_id, string calendar) \linebreak

Warning

This function is currently not documented; only the argument list is available.

icap_delete_event (PHP 4 >= 4.0.0)

Delete an event from an ICAP calendar

string **icap_delete_event** (int stream_id, int uid) \linebreak

icap_delete_event() deletes the calendar event specified by the *uid*.

Returns TRUE.

icap_fetch_event (PHP 4 >= 4.0.0)

Fetches an event from the calendar stream/

int **icap_fetch_event** (int stream_id, int event_id [, int options]) \linebreak

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_alarms (PHP 4 >= 4.0.0)

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) \linebreak

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

icap_list_events (PHP 4 >= 4.0.0)

Return a list of events between two given datetimes

array **icap_list_events** (int stream_id, int begin_date [, int end_date]) \linebreak

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_open (PHP 4 >= 4.0.0)

Opens up an ICAP connection

stream **icap_open** (string calendar, string username, string password, string options) \linebreak

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_rename_calendar (PHP 4 >= 4.0.0)

Rename a calendar

```
string icap_rename_calendar ( int stream_id, string old_name, string new_name) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

icap_reopen (PHP 4 >= 4.0.0)

Reopen ICAP stream to new calendar

```
int icap_reopen ( int stream_id, string calendar [, int options]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

icap_snooze (PHP 4 >= 4.0.0)

Snooze an alarm

```
string icap_snooze ( int stream_id, int uid) \linebreak
```

icap_snooze() turns on an alarm for a calendar event specified by the *uid*.

Returns **TRUE**.

icap_store_event (PHP 4 >= 4.0.0)

Store an event into an ICAP calendar

```
string icap_store_event ( int stream_id, object event) \linebreak
```

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.

XLI. iconv functions

This module contains an interface to the iconv library functions. To be able to use the functions defined in this module you must compile your PHP interpreter using the `--with-iconv` option. In order to do so, you must have `iconv()` function in standard C library or `libiconv` installed on your system. `libiconv` library is available from <http://www.gnu.org/software/libiconv/> (<http://www.gnu.org/software/libiconv/>).

`iconv` library function converts files between various encoded character sets. The supported character sets depend on `iconv()` implementation on your system. Note that `iconv()` function in some system is not work well as you expect. In this case, you should install `libiconv` library.

iconv (PHP 4 >= 4.0.5)

Convert string to requested character encoding

string **iconv** (string *in_charset*, string *out_charset*, string *str*) \linebreak

It converts the string *string* encoded in *in_charset* to the string encoded in *out_charset*. It returns the converted string or FALSE, if it fails.

Example 1. iconv() example:

```
echo iconv("ISO-8859-1", "UTF-8", "This is test.");
```

iconv_get_encoding (PHP 4 >= 4.0.5)

Get current setting for character encoding conversion

array **iconv_get_encoding** ([string *type*]) \linebreak

It returns the current settings of `ob_iconv_handler()` as array or FALSE in failure.

See also: `iconv_set_encoding()` and `ob_iconv_handler()`.

iconv_set_encoding (PHP 4 >= 4.0.5)

Set current setting for character encoding conversion

array **iconv_set_encoding** (string *type*, string *charset*) \linebreak

It changes the value of *type* to *charset* and returns TRUE in success or FALSE in failure.

Example 1. iconv_set_encoding() example:

```
iconv_set_encoding("internal_encoding", "UTF-8");  
iconv_set_encoding("output_encoding", "ISO-8859-1");
```

See also: `iconv_get_encoding()` and `ob_iconv_handler()`.

ob_iconv_handler (PHP 4 >= 4.0.5)

Convert character encoding as output buffer handler

array **ob_iconv_handler** (string contents, int status) \linebreak

It converts the string encoded in *internal_encoding* to *output_encoding*.

internal_encoding and *output_encoding* should be defined by `iconv_set_encoding()` or in configuration file.

Example 1. ob_iconv_handler() example:

```
ob_start("ob_iconv_handler"); // start output buffering
```

See also: `iconv_get_encoding()` and `iconv_set_encoding()`.

XLII. Image functions

You can use the image functions in PHP to get the size of JPEG, GIF, PNG, SWF, TIFF and JPEG2000 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.

If you have PHP compiled with `--enable-exif` you are able to work with information stored in headers of JPEG and TIFF images. These functions do not require GD library.

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://sunsite.unc.edu/pub/Linux/libs/graphics/>), and then add `--with-t1lib[=dir]`.

exif_imagetype (PHP 4 CVS only)

Determine the type of an image

```
int|false exif_imagetype ( string filename) \linebreak
```

exif_imagetype() reads the first bytes of an image and checks its signature. When a correct signature is found a constant will be returned otherwise the return value is `FALSE`. The return value is the same value that `getimagesize()` returns in index 2 but this function is much faster.

The following constants are defined: 1 = `IMAGETYPE_GIF`, 2 = `IMAGETYPE_JPG`, 3 = `IMAGETYPE_PNG`, 4 = `IMAGETYPE_SWF`, 5 = `IMAGETYPE_PSD`, 6 = `IMAGETYPE_BMP`, 7 = `IMAGETYPE_TIFF_II` (intel byte order), 8 = `IMAGETYPE_TIFF_MM` (motorola byte order), 9 = `IMAGETYPE_JPC`, 10 = `IMAGETYPE_JP2`, 11 = `IMAGETYPE_JPX`.

This function can be used to avoid calls to other exif functions with unsupported file teypes or in conjunction with `$_SERVER['HTTP_ACCEPT']` to check whether or not the viewer is able to see a specific image in his browser.

Note: This function is only available in PHP 4 compiled using `--enable-exif`.

This function does not require the GD image library.

See also `getimagesize()`.

exif_read_data (PHP 4 >= 4.2.0)

Read the EXIF headers from JPEG or TIFF

```
array exif_read_data ( string filename [, string sections [, bool arrays [, bool thumbnail]]]) \linebreak
```

The **exif_read_data()** function reads the EXIF headers from a JPEG or TIFF image file. It returns an associative array where the indexes are the header names and the values are the values associated with those headers. If no data can be returned the result is `FALSE`.

filename is the name of the file to read. This cannot be a url.

sections a comma separated lsit of sections that need to be present in file to produce a result array.

FILE	FileName, FileSize, FileDateTime, SectionsFound
COMPUTED	html, Width, Height, IsColor and some more if available.
ANY_TAG	Any information that has a Tag e.g. IFD0, EXIF, ...
IFD0	All tagged data of IFD0. In normal imagefiles this contains image size and so forth.

THUMBNAIL	A file is supposed to contain a thumbnail if it has a second IFD. All tagged information about the embedded thumbnail is stored in this section.
COMMENT	Comment headers of JPEG images.
EXIF	The EXIF section is a sub section of IFD0. It contains more detailed information about an image. Most of these entries are digital camera related.

arrays specifies whether or not each section becomes an array. The sections *FILE*, *COMPUTED* and *THUMBNAIL* always become arrays as they may contain values whose names are conflict with other sections.

tthumbnail whether or not to read the thumbnail itself and not only its tagged data.

Note: Exif headers tend to be present in JPEG/TIFF 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. `exif_read_data()` example

```
<?php
echo "test1.jpg:<br>\n";
$exif = exif_read_data ('tests/test1.jpg', 'IFD0');
echo $exif===false ? "No header data found.<br>\n" : "Image contains headers<br>";
$exif = exif_read_data ('tests/test2.jpg', 0, true);
echo "test2.jpg:<br>\n";
foreach($exif as $key=>$section) {
    foreach($section as $name=>$val) {
        echo "$key.$name: $val<br>\n";
    }
}
}??>
```

The first call fails because the image has no header information.

```
test1.jpg:
No header data found.
test2.jpg:
FILE.FileName: test2.jpg
FILE.FileDateTime: 1017666176
FILE.FileSize: 1240
FILE.FileType: 2
FILE.SectionsFound: ANY_TAG, IFD0, THUMBNAIL, COMMENT
COMPUTED.html: width="1" height="1"
```



```

COMPUTED.Height: 1
COMPUTED.Width: 1
COMPUTED.IsColor: 1
COMPUTED.ByteOrderMotorola: 1
COMPUTED.UserComment: Exif test image.
COMPUTED.UserCommentEncoding: ASCII
COMPUTED.Copyright: Photo (c) M.Boerger, Edited by M.Boerger.
COMPUTED.Copyright.Photographer: Photo (c) M.Boerger
COMPUTED.Copyright.Editor: Edited by M.Boerger.
IFD0.Copyright: Photo (c) M.Boerger
IFD0.UserComment: ASCII
THUMBNAIL.JPEGInterchangeFormat: 134
THUMBNAIL.JPEGInterchangeFormatLength: 523
COMMENT.0: Comment #1.
COMMENT.1: Comment #2.
COMMENT.2: Comment #3end

```

Note: If the image contains any IFD0 data then COMPUTED contains the entry `ByteOrderMotorola` which is 0 for little-endian (intel) and 1 for big-endian (motorola) byte order. This was added in PHP 4.3.

When an Exif header contains a Copyright note this itself can contain two values. As the solution is inconsistent in the Exif 2.10 standard the COMPUTED section will return both entries `Copyright.Photographer` and `Copyright.Editor` while the IFD0 sections contains the byte array with the NULL character that splits both entries. Or just the first entry if the datatype was wrong (normal behaviour of Exif). The COMPUTED will contain also an entry `Copyright` Which is either the original copyright string or it is a comma separated list of photo and editor copyright.

Note: The tag `UserComment` has the same problem as the `Copyright` tag. It can store two values first the encoding used and second the value itself. If so the IFD section only contains the encoding or a byte array. The COMPUTED section will store both in the entries `UserCommentEncoding` and `UserComment`. The entry `UserComment` is available in both cases so it should be used in preference to the value in IFD0 section.

If the user comment uses Unicode or JIS encoding and the module `mbstring` is available this encoding will automatically changed according to the `exif.ini` settings. This was added in PHP 4.3.

Note: Height and Width are computed the same way `getimagesize()` does so their values must not be part of any header returned. Also `html` is a height/width text string to be used inside normal HTML.

Note: Starting from PHP 4.3 the function can read all embedded IFD data including arrays (returned as such). Also the size of an embedded thumbnail is returned in *THUMBNAIL* subarray and the function **exif_read_data()** can return thumbnails in TIFF format. Last but not least there is no longer a maximum length for returned values (not until memory limit is reached).

Note: This function is only available in PHP 4 compiled using `--enable-exif`. Its functionality and behaviour has changed in PHP 4.2. Earlier versions are very unstable.

Since PHP 4.3 user comment can automatically change encoding if PHP 4 was compiled using `--enable-mbstring`.

This function does not require the GD image library.

See also `exif_thumbnail()` and `getimagesize()`.

exif_thumbnail (PHP 4 >= 4.2.0)

Retrieve the embedded thumbnail of a TIFF or JPEG image

string **exif_thumbnail** (string filename [, int &width [, int &height]]) \linebreak

exif_thumbnail() reads the embedded thumbnail of a TIFF or JPEG image. If the image contains no thumbnail `FALSE` will be returned.

Both parameters *width* and *height* are available since PHP 4.3 and return the size of the thumbnail. It is possible that **exif_thumbnail()** cannot create an image but determine its size. In this case the return value is `FALSE` but *width* and *height* are set.

Starting from version PHP 4.3 the function **exif_thumbnail()** can return thumbnails in TIFF format.

Note: This function is only available in PHP 4 compiled using `--enable-exif`. Its functionality and behaviour has changed in PHP 4.2

This function does not require the GD image library.

See also `exif_read_data()`.

getimagesize (PHP 3, PHP 4 >= 4.0.0)

Get the size of an image

array **getimagesize** (string filename [, array imageinfo]) \linebreak

The **getimagesize()** function will determine the size of any GIF, JPG, PNG, SWF, PSD, TIFF or BMP 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, 5 = PSD, 6 = BMP, 7 = TIFF(intel byte order), 8 = TIFF(motorola byte order, 9 = JPC, 10 = JP2, 11 = JPX. Index 3 is a text string with the correct height="yyy" width="xxx" string that can be used directly in an `IMG` tag.

Example 1. getimagesize (file)

```
<?php
$size = getimagesize ("img/flag.jpg");
echo "<img src=\"img/flag.jpg\" {$size[3]}>";
?>
```

Example 2. getimagesize (URL)

```
<?php $size = getimagesize ("http://www.example.com/gifs/logo.gif"); ?>
```

With JPG images, two extras index are returned : `channel` and `bits`. `channel` will be 3 for RGB pictures, and 4 for CMYK pictures. `bits` is the number of bits for each color.

If accessing the `filename` image is impossible, or if it isn't a valid picture, **getimagesize()** will return `NULL` and generate a warning.

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 embedd text information in images. A very common one in 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: TIFF support was added in PHP 4.2. JPEG2000 support will be added in PHP 4.3.

This function does not require the GD image library.

See also `exif_imagetype()`, `exif_read_data()` and `exif_thumbnail()`.

URL support was added in PHP 4.0.5

image2wbmp (PHP 4 >= 4.0.5)

Output image to browser or file

int **image2wbmp** (resource image [, string filename [, int threshold]]) \linebreak

image2wbmp() creates the WBMP file in filename from the image *image*. The *image* argument is the return from `imagecreate()`.

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 if PHP was compiled against GD-1.8 or later.

See also `imagewbmp()`.

imagealphablending (PHP 4 >= 4.0.6)

Set the blending mode for an image

int **imagealphablending** (resource image, bool blendmode) \linebreak

imagealphablending() allows for two different modes of drawing on truecolor images. In blending mode, the alpha channel component of the color supplied to all drawing function, such as `imagepixel()` determines how much of the underlying color should be allowed to shine through. As a result, gd automatically blends the existing color at that point with the drawing color, and stores the result in the image. The resulting pixel is opaque. In non-blending mode, the drawing color is copied literally with its alpha channel information, replacing the destination pixel. Blending mode is not available when drawing on palette images. If *blendmode* is `TRUE`, then blending mode is enabled, otherwise disabled.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1

imagearc (PHP 3, PHP 4 >= 4.0.0)

Draw a partial ellipse

int **imagearc** (resource image, int cx, int cy, int w, int h, int s, int e, int col) \linebreak

imagearc() draws a partial ellipse centered at cx , cy (top left is 0, 0) in the image represented by *image*. 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. 0° is located at the three-o'clock position, and the arc is drawn counter-clockwise.

See also `imageellipse()`, `imagefilledellipse()`, and `imagefilledarc()`.

imagechar (PHP 3, PHP 4 >= 4.0.0)

Draw a character horizontally

int **imagechar** (resource image, int font, int x, int y, string c, int col) \linebreak

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 >= 4.0.0)

Draw a character vertically

int **imagecharup** (resource image, int font, int x, int y, string c, int col) \linebreak

imagecharup() draws the character c vertically in the image identified by *image* 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 >= 4.0.0)

Allocate a color for an image

int **imagecolorallocate** (resource image, int red, int green, int blue) \linebreak

imagecolorallocate() returns a color identifier representing the color composed of the given RGB components. The *im* argument is the return from the `imagecreate()` function. *red*, *green* and *blue* are the values of the red, green and blue component of the requested color respectively. These parameters are integers between 0 and 255. **imagecolorallocate()** must be called to create each color that is to be used in the image represented by *image*.

```
$white = imagecolorallocate ($im, 255, 255, 255);
$black = imagecolorallocate ($im, 0, 0, 0);
```

Returns -1 if the allocation failed.

imagecolorat (PHP 3, PHP 4 >= 4.0.0)

Get the index of the color of a pixel

```
int imagecolorat ( resource image, int x, int y) \linebreak
```

Returns the index of the color of the pixel at the specified location in the image specified by *image*.

See also `imagecolorset()` and `imagecolorsforindex()`.

imagecolorclosest (PHP 3, PHP 4 >= 4.0.0)

Get the index of the closest color to the specified color

```
int imagecolorclosest ( resource image, int red, int green, int blue) \linebreak
```

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()`.

imagecolorclosestalpha (PHP 4 >= 4.0.6)

Get the index of the closest color to the specified color + alpha

```
int imagecolorclosestalpha ( resource image, int red, int green, int blue, int alpha) \linebreak
```

Returns the index of the color in the palette of the image which is "closest" to the specified RGB value and *alpha* level.

See also `imagecolorexactalpha()`.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1

`imagecolorclosesthwb` (PHP 4)

Get the index of the color which has the hue, white and blackness nearest to the given color

`int imagecolorclosesthwb (resource image, int red, int green, int blue) \linebreak`

Warning

This function is currently not documented; only the argument list is available.

`imagecolordeallocate` (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

De-allocate a color for an image

`int imagecolordeallocate (resource image, int color) \linebreak`

The `imagecolordeallocate()` function de-allocates a color previously allocated with the `imagecolorallocate()` function.

```
$white = imagecolorallocate ($im, 255, 255, 255);  
imagecolordeallocate ($im, $white);
```

`imagecolorexact` (PHP 3, PHP 4 >= 4.0.0)

Get the index of the specified color

`int imagecolorexact (resource image, int red, int green, int blue) \linebreak`

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()`.

imagecolorexactalpha (PHP 4 >= 4.0.6)

Get the index of the specified color + alpha

int **imagecolorexactalpha** (resource image, int red, int green, int blue, int alpha) \linebreak

Returns the index of the specified color+alpha in the palette of the image.

If the color does not exist in the image's palette, -1 is returned.

See also `imagecolorclosestalpha()`.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

imagecolorresolve (PHP 3 >= 3.0.2, PHP 4 >= 4.0.0)

Get the index of the specified color or its closest possible alternative

int **imagecolorresolve** (resource image, int red, int green, int blue) \linebreak

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()`.

imagecolorresolvealpha (PHP 4 >= 4.0.6)

Get the index of the specified color + alpha or its closest possible alternative

int **imagecolorresolvealpha** (resource image, int red, int green, int blue, int alpha) \linebreak

This function is guaranteed to return a color index for a requested color, either the exact color or the closest possible alternative.

See also `imagecolorclosestalpha()`.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1

imagecolorset (PHP 3, PHP 4 >= 4.0.0)

Set the color for the specified palette index

bool **imagecolorset** (resource image, int index, int red, int green, int blue) \linebreak

This sets the specified index in the palette to the specified color. This is useful for creating flood-fill-like effects in paletted images without the overhead of performing the actual flood-fill.

See also `imagecolorat()`.

imagecolorsforindex (PHP 3, PHP 4 >= 4.0.0)

Get the colors for an index

array **imagecolorsforindex** (resource image, int index) \linebreak

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 >= 4.0.0)

Find out the number of colors in an image's palette

int **imagecolorstotal** (resource image) \linebreak

This returns the number of colors in the specified image's palette.

See also `imagecolorat()` and `imagecolorsforindex()`.

imagecolortransparent (PHP 3, PHP 4 >= 4.0.0)

Define a color as transparent

int **imagecolortransparent** (resource image [, int color]) \linebreak

imagecolortransparent() sets the transparent color in the *image* image to *color*. *image* is the image identifier returned by `imagecreate()` and *color* is a color identifier returned by `imagecolorallocate()`.

Note: The transparent color is a property of the image, transparency is not a property of the color. Once you have a set a color to be the transparent color, any regions of the image in that color that were drawn previously will be transparent.

The identifier of the new (or current, if none is specified) transparent color is returned.

imagecopy (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Copy part of an image

int imagecopy (resource *dst_im*, resource *src_im*, int *dst_x*, int *dst_y*, int *src_x*, int *src_y*, int *src_w*, int *src_h*)
 \linebreak

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*.

imagecopymerge (PHP 4)

Copy and merge part of an image

int imagecopymerge (resource *dst_im*, resource *src_im*, int *dst_x*, int *dst_y*, int *src_x*, int *src_y*, int *src_w*, int *src_h*, int *pct*)
 \linebreak

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*. The two images will be merged according to *pct* which can range from 0 to 100. When *pct* = 0, no action is taken, when 100 this function behaves identically to `imagecopy()`.

Note: This function was added in PHP 4.0.6

imagecopymergegray (PHP 4 >= 4.0.6)

Copy and merge part of an image with gray scale

int imagecopymergegray (resource *dst_im*, resource *src_im*, int *dst_x*, int *dst_y*, int *src_x*, int *src_y*, int *src_w*, int *src_h*, int *pct*)
 \linebreak

imagecopymergegray() 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*. The two images will be merged according to *pct* which can range from 0 to 100. When *pct* = 0, no action is taken, when 100 this function behaves identically to `imagecopy()`.

This function is identical to `imagecopymerge()` except that when merging it preserves the hue of the source by converting the destination pixels to gray scale before the copy operation.

Note: This function was added in PHP 4.0.6

imagecopyresampled (PHP 4 >= 4.0.6)

Copy and resize part of an image with resampling

int **imagecopyresampled** (resource *dst_im*, resource *src_im*, int *dstX*, int *dstY*, int *srcX*, int *srcY*, int *dstW*, int *dstH*, int *srcW*, int *srcH*) \linebreak

imagecopyresampled() copies a rectangular portion of one image to another image, smoothly interpolating pixel values so that, in particular, reducing the size of an image still retains a great deal of clarity. *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.

See also `imagecopyresized()`.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

imagecopyresized (PHP 3, PHP 4 >= 4.0.0)

Copy and resize part of an image

int **imagecopyresized** (resource *dst_im*, resource *src_im*, int *dstX*, int *dstY*, int *srcX*, int *srcY*, int *dstW*, int *dstH*, int *srcW*, int *srcH*) \linebreak

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.

See also `imagecopyresampled()`.

imagecreate (PHP 3, PHP 4 >= 4.0.0)

Create a new palette based image

resource **imagecreate** (int *x_size*, int *y_size*) \linebreak

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);
?>
```

imagecreatefromgd (PHP 4 >= 4.1.0)

Create a new image from GD file or URL

resource **imagecreatefromgd** (string *filename*) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagecreatefromgd2 (PHP 4 >= 4.1.0)

Create a new image from GD2 file or URL

resource **imagecreatefromgd2** (string *filename*) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagecreatefromgd2part (PHP 4 >= 4.1.0)

Create a new image from a given part of GD2 file or URL

resource **imagecreatefromgd2part** (string filename, int srcX, int srcY, int width, int height) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagecreatefromgif (PHP 3, PHP 4 >= 4.0.0)

Create a new image from file or URL

resource **imagecreatefromgif** (string filename) \linebreak

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@zysms.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 >= 4.0.0)

Create a new image from file or URL

resource **imagecreatefromjpeg** (string filename) \linebreak

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@zysys.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 >= 4.0.0)

Create a new image from file or URL

resource **imagecreatefrompng** (string filename) \linebreak

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@zysys.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;
}
```

imagecreatefromstring (PHP 4 >= 4.0.4)

Create a new image from the image stream in the string

resource **imagecreatefromstring** (string image) \linebreak

imagecreatefromstring() returns an image identifier representing the image obtained from the given string.

imagecreatefromwbmp (PHP 4)

Create a new image from file or URL

resource **imagecreatefromwbmp** (string filename) \linebreak

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@zysys.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;
}

```

Note: WBMP support is only available if PHP was compiled against GD-1.8 or later.

imagecreatefromxbm (PHP 4)

Create a new image from file or URL

resource **imagecreatefromxbm** (string filename) \linebreak

imagecreatefromxbm() returns an image identifier representing the image obtained from the given filename.

imagecreatefromxpm (PHP 4)

Create a new image from file or URL

resource **imagecreatefromxpm** (string filename) \linebreak

imagecreatefromxpm() returns an image identifier representing the image obtained from the given filename.

imagecreatetruecolor (PHP 4 >= 4.0.6)

Create a new true color image

resource **imagecreatetruecolor** (int x_size, int y_size) \linebreak

imagecreatetruecolor() returns an image identifier representing a black image of size *x_size* by *y_size*.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

imagedashedline (PHP 3, PHP 4 >= 4.0.0)

Draw a dashed line

int **imagedashedline** (resource image, int x1, int y1, int x2, int y2, int col) \linebreak

This function is deprecated. Use combination of imagesetstyle() and imageline() instead.

imagedestroy (PHP 3, PHP 4 >= 4.0.0)

Destroy an image

int **imagedestroy** (resource image) \linebreak

imagedestroy() frees any memory associated with image *image*. *image* is the image identifier returned by the imagecreate() function.

imageellipse (PHP 4 >= 4.0.6)

Draw an ellipse

int **imageellipse** (resource image, int cx, int cy, int w, int h, int col) \linebreak

imageellipse() draws an ellipse centered at *cx*, *cy* (top left is 0, 0) in the image represented by *image*. *W* and *h* specifies the ellipse's width and height respectively. The color of the ellipse is specified by *color*.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.2 or later

See also imagearc().

imagefill (PHP 3, PHP 4 >= 4.0.0)

Flood fill

int **imagefill** (resource image, int x, int y, int col) \linebreak

imagefill() performs a flood fill starting at coordinate x, y (top left is 0, 0) with color col in the image $image$.

imagefilledarc (PHP 4 >= 4.0.6)

Draw a partial ellipse and fill it

int **imagefilledarc** (resource image, int cx, int cy, int w, int h, int s, int e, int col, int style) \linebreak

imagefilledarc() draws a partial ellipse centered at cx, cy (top left is 0, 0) in the image represented by $image$. 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. $style$ is a bitwise OR of the following possibilities:

1. IMG_ARC_PIE
2. IMG_ARC_CHORD
3. IMG_ARC_NOFILL
4. IMG_ARC_EDGED

IMG_ARC_PIE and IMG_ARC_CHORD are mutually exclusive; IMG_ARC_CHORD just connects the starting and ending angles with a straight line, while IMG_ARC_PIE produces a rounded edge. IMG_ARC_NOFILL indicates that the arc or chord should be outlined, not filled. IMG_ARC_EDGED, used together with IMG_ARC_NOFILL, indicates that the beginning and ending angles should be connected to the center - this is a good way to outline (rather than fill) a 'pie slice'.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1

imagefilledellipse (PHP 4 >= 4.0.6)

Draw a filled ellipse

int **imagefilledellipse** (resource image, int cx, int cy, int w, int h, int col) \linebreak

imagefilledellipse() draws an ellipse centered at cx, cy (top left is 0, 0) in the image represented by $image$. W and h specifies the ellipse's width and height respectively. The ellipse is filled using $color$

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

See also imagefilledarc().

imagefilledpolygon (PHP 3, PHP 4 >= 4.0.0)

Draw a filled polygon

int **imagefilledpolygon** (resource image, array points, int num_points, int col) \linebreak

imagefilledpolygon() creates a filled polygon in image *image*. *points* is a PHP array containing the polygon's vertices, ie. points[0] = x0, points[1] = y0, points[2] = x1, points[3] = y1, etc. *num_points* is the total number of vertices.

imagefilledrectangle (PHP 3, PHP 4 >= 4.0.0)

Draw a filled rectangle

int **imagefilledrectangle** (resource image, int x1, int y1, int x2, int y2, int col) \linebreak

imagefilledrectangle() creates a filled rectangle of color *col* in image *image* starting at upper left coordinates *x1*, *y1* and ending at bottom right coordinates *x2*, *y2*. 0, 0 is the top left corner of the image.

imagefilltoborder (PHP 3, PHP 4 >= 4.0.0)

Flood fill to specific color

int **imagefilltoborder** (resource image, int x, int y, int border, int col) \linebreak

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 >= 4.0.0)

Get font height

int **imagefontheight** (int font) \linebreak

Returns the pixel height of a character in the specified font.

See also `imagefontwidth()` and `imageloadfont()`.

imagefontwidth (PHP 3, PHP 4 >= 4.0.0)

Get font width

int **imagefontwidth** (int font) \linebreak

Returns the pixel width of a character in font.

See also `imagefontheight()` and `imageloadfont()`.

imageftbbox (PHP 4 >= 4.1.0)

Give the bounding box of a text using fonts via freetype2

array **imageftbbox** (int size, int angle, string font_file, string text [, array extrainfo]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagefttext (PHP 4 >= 4.1.0)

Write text to the image using fonts using FreeType 2

array **imagefttext** (resource image, int size, int angle, int x, int y, int col, string font_file, string text [, array extrainfo]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagegammacorrect (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Apply a gamma correction to a GD image

int **imagegammacorrect** (resource image, float inputgamma, float outputgamma) \linebreak

The **imagegammacorrect()** function applies gamma correction to a gd image stream (*image*) given an input gamma, the parameter *inputgamma* and an output gamma, the parameter *outputgamma*.

imagegd (PHP 4 >= 4.1.0)

Output GD image to browser or file

int **imagegd** (resource image [, string filename]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagegd2 (PHP 4 >= 4.1.0)

Output GD2 image to browser or file

int **imagegd2** (resource image [, string filename]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imagegif (PHP 3, PHP 4 >= 4.0.0)

Output image to browser or file

int **imagegif** (resource image [, string filename]) \linebreak

imagegif() creates the GIF file in filename from the image *image*. The *image* 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 ($im);
}
elseif (function_exists("imagebmp")) {
    header ("Content-type: image/vnd.wap.wbmp");
    imagebmp ($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()`, `imagebmp()`, `imagejpeg()`, `imagetypes()`.

imageinterlace (PHP 3, PHP 4 >= 4.0.0)

Enable or disable interlace

int imageinterlace (resource image [, int interlace]) \linebreak

imageinterlace() turns the interlace bit on or off. If interlace is 1 the image will be interlaced, and if interlace is 0 the interlace bit is turned off.

If the interlace bit is set and the image is used as a JPEG image, the image is created as a progressive JPEG.

This function returns whether the interlace bit is set for the image.

imagejpeg (PHP 3 >= 3.0.16, PHP 4 >= 4.0.0)

Output image to browser or file

int **imagejpeg** (resource image [, string filename [, int quality]]) \linebreak

imagejpeg() creates the JPEG file in filename from the image *image*. The *image* 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 if PHP was compiled against GD-1.8 or later.

quality is optional, and ranges from 0 (worst quality, smaller file) to 100 (best quality, biggest file). The default is the default IJG quality value (about 75).

If you want to output Progressive JPEGs, you need to set interlacing on with `imageinterlace()`.

See also `imagepng()`, `imagegif()`, `imagewbmp()`, `imageinterlace()` and `imagetypes()`.

imageline (PHP 3, PHP 4 >= 4.0.0)

Draw a line

int **imageline** (resource image, int x1, int y1, int x2, int y2, int col) \linebreak

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 >= 4.0.0)

Load a new font

int **imageloadfont** (string file) \linebreak

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()`.

imagepalettecopy (PHP 4)

Copy the palette from one image to another

int **imagepalettecopy** (resource destination, resource source) \linebreak

imagepalettecopy() copies the palette from the *source* image to the *destination* image.

imagepng (PHP 3 >= 3.0.13, PHP 4 >= 4.0.0)

Output a PNG image to either the browser or a file

int **imagepng** (resource image [, string filename]) \linebreak

The **imagepng()** outputs a GD image stream (*image*) 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()`.

imagepolygon (PHP 3, PHP 4 >= 4.0.0)

Draw a polygon

int **imagepolygon** (resource image, array points, int num_points, int col) \linebreak

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 >= 4.0.0)

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]]]) \linebreak

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()`.

ImagePSCopyFont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Make a copy of an already loaded font for further modification

`int imagepscopyfont (int fontindex) \linebreak`

Use this function if you need make further modifications to the font, for example extending/condensing, slanting it or changing it's character encoding vector, but need to keep the original along as well. Note that the font you want to copy must be one obtained using **ImagePSLoadFont()**, not a font that is itself a copied one. You can although make modifications to it before copying.

If you use this function, you *must* free the fonts obtained this way yourself and in reverse order. Otherwise your script *will* hang.

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 **ImagePSLoadFont()**.

imagepsencodefont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Change the character encoding vector of a font

`int imagepsencodefont (int font_index, string encodingfile) \linebreak`

Loads a character encoding vector from 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.

imagepsextendfont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Extend or condense a font

`bool imagepsextendfont (int font_index, float extend) \linebreak`

Extend or condense a font (`font_index`), if the value of the `extend` parameter is less than one you will be condensing the font.

imagepsfreefont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Free memory used by a PostScript Type 1 font

void **imagepsfreefont** (int fontindex) \linebreak

See also imagepsloadfont().

imagepsloadfont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Load a PostScript Type 1 font from file

int **imagepsloadfont** (string filename) \linebreak

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, which you cannot read directly, while the output type is image.

```
<?php
header ("Content-type: image/jpeg");
$im = imagecreate (350, 45);
$black = imagecolorallocate ($im, 0, 0, 0);
$white = imagecolorallocate ($im, 255, 255, 255);
$font = imagepsloadfont ("bchbi.pfb"); // or locate your .pfb files on your machine
imagepstext ($im, "Testing... It worked!", $font, 32, $white, $black, 32, 32);
imagepsfreefont ($font);
imagejpeg ($im, "", 100); //for best quality...your mileage may vary
imagedestroy ($im);
?>
```

See also imagepsfreefont().

imagepslantfont (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Slant a font

bool **imagepslantfont** (int font_index, float slant) \linebreak

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 >= 4.0.0)

To draw a text string over an image using PostScript Type1 fonts

array **imagepstext** (resource image, string text, int font, int size, int foreground, int background, int x, int y [, int space [, int tightness [, float angle [, int antialias_steps]]]]) \linebreak

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 >= 4.0.0)

Draw a rectangle

int **imagerectangle** (resource image, int x1, int y1, int x2, int y2, int col) \linebreak

imagerectangle() creates a rectangle of color *col* in image *image* 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.

imagesetbrush (PHP 4 >= 4.0.6)

Set the brush image for line drawing

int **imagesetbrush** (resource image, resource brush) \linebreak

imagesetbrush() sets the brush image to be used by all line drawing functions (such as `imageline()` and `imagepolygon()`) when drawing with the special colors `IMG_COLOR_BRUSHED` or `IMG_COLOR_STYLED` or `IMG_COLOR_STYLEDBRUSHED`.

Note: You need not take special action when you are finished with a brush, but if you destroy the brush image, you must not use the `IMG_COLOR_BRUSHED` or `IMG_COLOR_STYLED` colors until you have set a new brush image!

Note: This function was added in PHP 4.0.6

imagesetpixel (PHP 3, PHP 4 >= 4.0.0)

Set a single pixel

int **imagesetpixel** (resource image, int x, int y, int col) \linebreak

imagesetpixel() draws a pixel at *x*, *y* (top left is 0, 0) in image *image* of color *col*.

See also `imagecreate()` and `imagecolorallocate()`.

imagesetstyle (PHP 4 >= 4.0.6)

Set the style for line drawing

int **imagesetstyle** (resource image, array style) \linebreak

imagesetstyle() sets the style to be used by all line drawing functions (such as `imageline()` and `imagepolygon()`) when drawing with the special color `IMG_COLOR_STYLED` or lines of images with color `IMG_COLOR_STYLED` or `IMG_COLOR_STYLEDBRUSHED`.

The *style* parameter is an array of pixels. Following example script draws a dashed line from upper left to lower right corner of the canvas:

Example 1. imagesetstyle

```

<?php
header ("Content-type: image/png");
$im = imagecreate (100, 100);
$w  = imagecolorallocate ($im, 255, 255, 255);
$red = imagecolorallocate ($im, 255, 0, 0);

/* Draw a dashed line, 5 red pixels, 5 white pixels */
$style = array ($red,$red,$red,$red,$red,$w,$w,$w,$w,$w);
imagesetstyle ($im, $style);
imageline ($im, 0, 0, 100, 100, IMG_COLOR_STYLED);

/* Draw a line of happy faces using imagesetbrush() with imagesetstyle */
$style = array ($w,$w,$w,$w,$w,$w,$w,$w,$w,$w,$w,$w,$red);
imagesetstyle ($im, $style);

$brush = imagecreatefrompng ("http://www.libpng.org/pub/png/images/smile.happy.png");
imagecolortransparent ($brush, $w);
imagesetbrush ($im, $brush);
imageline ($im, 100, 0, 0, 100, IMG_COLOR_STYLEDBRUSHED);

imagepng ($im);
imagedestroy ($im);
?>

```

See also `imagesetbrush()`, `imageline()`.

Note: This function was added in PHP 4.0.6

imagesetthickness (PHP 4 >= 4.0.6)

Set the thickness for line drawing

void **imagesetthickness** (resource image, int thickness) \linebreak

imagesetthickness() sets the thickness of the lines drawn when drawing rectangles, polygons, ellipses etc. etc. to *thickness* pixels.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

imagesttile (PHP 4 >= 4.0.6)

Set the tile image for filling

int **imagesttile** (resource image, resource tile) \linebreak

imagesttile() sets the tile image to be used by all region filling functions (such as `imagefill()` and `imagefilledpolygon()`) when filling with the special color `IMG_COLOR_TILED`.

A tile is an image used to fill an area with a repeated pattern. Any GD image can be used as a tile, and by setting the transparent color index of the tile image with `imagecolortransparent()`, a tile allows certain parts of the underlying area to shine through can be created.

Note: You need not take special action when you are finished with a tile, but if you destroy the tile image, you must not use the `IMG_COLOR_TILED` color until you have set a new tile image!

Note: This function was added in PHP 4.0.6

imagestring (PHP 3, PHP 4 >= 4.0.0)

Draw a string horizontally

int **imagestring** (resource image, int font, int x, int y, string s, int col) \linebreak

imagestring() draws the string *s* in the image identified by *image* 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 >= 4.0.0)

Draw a string vertically

int **imagestringup** (resource image, int font, int x, int y, string s, int col) \linebreak

imagestringup() draws the string *s* vertically in the image identified by *image* 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 >= 4.0.0)

Get image width

int **imagesx** (resource image) \linebreak

imagesx() returns the width of the image identified by *image*.

See also `imagecreate()` and `imagesy()`.

imagesy (PHP 3, PHP 4 >= 4.0.0)

Get image height

int **imagesy** (resource image) \linebreak

imagesy() returns the height of the image identified by *image*.

See also `imagecreate()` and `imagesx()`.

imagetruecolortopalette (PHP 4 >= 4.0.6)

Convert a true color image to a palette image

void **imagetruecolortopalette** (resource image, bool dither, int ncolors) \linebreak

imagetruecolortopalette() converts a truecolor image to a palette image. The code for this function was originally drawn from the Independent JPEG Group library code, which is excellent. The code has been modified to preserve as much alpha channel information as possible in the resulting palette, in addition to preserving colors as well as possible. This does not work as well as might be hoped. It is usually best to simply produce a truecolor output image instead, which guarantees the highest output quality.

dither indicates if the image should be dithered - if it is `TRUE` then dithering will be used which will result in a more speckled image but with better color approximation.

ncolors sets the maximum number of colors that should be retained in the palette.

Note: This function was added in PHP 4.0.6 and requires GD 2.0.1 or later

imagettfbbox (PHP 3>= 3.0.1, PHP 4 >= 4.0.0)

Give the bounding box of a text using TrueType fonts

array **imagettfbbox** (int size, int angle, string fontfile, string text) \linebreak

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.) Depending on which version of the GD library that PHP is using, it may attempt to search for files that do not begin with a leading '/' by appending '.ttf' to the filename and searching along a library-defined font path.

angle

Angle in degrees in which *text* will be measured.

imagettfbbox() returns an array with 8 elements representing four points making the bounding box of the text:

0	lower left corner, X position
1	lower left corner, Y position
2	lower right corner, X position
3	lower right corner, Y position
4	upper right corner, X position
5	upper right corner, Y position
6	upper left corner, X position
7	upper left corner, Y position

The points are relative to the *text* regardless of the angle, so "upper left" means in the top left-hand corner seeing the text horizontally.

This function requires both the GD library and the FreeType library.

See also `imagettftext()`.

imagettftext (PHP 3, PHP 4 >= 4.0.0)

Write text to the image using TrueType fonts

array **imagettftext** (resource image, int size, int angle, int x, int y, int col, string fontfile, string text) \linebreak

imagettftext() draws the string *text* in the image identified by *image*, 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*. Depending on which version of the GD library that PHP is using, when *fontfile* does not begin with a leading '/', '.ttf' will be appended to the filename and the the library will attempt to search for that filename along a library-defined font path.

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 lower left, lower right, upper right, upper 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 3 CVS only, PHP 4 >= 4.0.2)

Return the image types supported by this PHP build

int **imagetypes** (void) \linebreak

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";
}
?>
```

imagewbmp (PHP 3>= 3.0.15, PHP 4)

Output image to browser or file

int **imagewbmp** (resource image [, string filename [, int foreground]]) \linebreak

imagewbmp() creates the WBMP file in filename from the image *image*. The *image* 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 if PHP was compiled against GD-1.8 or later.

Using the optional *foreground* parameter, you can set the foreground color. Use an identifier obtained from `imagecolorallocate()`. The default foreground color is black.

See also `image2wbmp()`, `imagepng()`, `imagegif()`, `imagejpeg()`, `imagetypes()`.

iptcembed (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Embed binary IPTC data into a JPEG image

array **iptcembed** (string iptcdata, string jpeg_file_name [, int spool]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

jpeg2wbmp (PHP 4 >= 4.0.5)

Convert JPEG image file to WBMP image file

int **jpeg2wbmp** (string jpegname, string wbmpname, int d_height, int d_width, int threshold) \linebreak

Converts the *jpegname* JPEG file to WBMP format, and saves it as *wbmpname*. With the *d_height* and *d_width* you specify the height and width of the destination image.

Note: WBMP support is only available if PHP was compiled against GD-1.8 or later.

See also png2wbmp().

png2wbmp (PHP 4 >= 4.0.5)

Convert PNG image file to WBMP image file

int **png2wbmp** (string pngname, string wbmpname, int d_height, int d_width, int threshold) \linebreak

Converts the *pngname* PNG file to WBMP format, and saves it as *wbmpname*. With the *d_height* and *d_width* you specify the height and width of the destination image.

Note: WBMP support is only available if PHP was compiled against GD-1.8 or later.

See also jpeg2wbmp().

read_exif_data (PHP 4)

Reads header information stored in TIFF and JPEG images

array **exif_read_data** (string filename, string sections, bool arrays, bool thumbnail) \linebreak

Note: The **read_exif_data()** function is an alias for `exif_read_data()`.

See also `exif_thumbnail()`.

XLIII. 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/*.h` to `/usr/local/include` or some other directory in your include path.

Note: Depending how the `c-client` was configured, you might also need to add `--with-imap-ssl=/path/to/openssl/` and/or `--with-kerberos` into the PHP configure line.

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:

- RFC2821 (<http://www.faqs.org/rfcs/rfc2821.html>): Simple Mail Transfer Protocol (SMTP).
- RFC2822 (<http://www.faqs.org/rfcs/rfc2822.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_8bit (PHP 3, PHP 4 >= 4.0.0)

Convert an 8bit string to a quoted-printable string

string **imap_8bit** (string string) \linebreak

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_alerts (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

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) \linebreak

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_append (PHP 3, PHP 4 >= 4.0.0)

Append a string message to a specified mailbox

int **imap_append** (int imap_stream, string mbox, string message [, string flags]) \linebreak

Returns `TRUE` on success, `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 >= 4.0.0)

Decode BASE64 encoded text

string **imap_base64** (string text) \linebreak

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_binary (PHP 3>= 3.0.2, PHP 4 >= 4.0.0)

Convert an 8bit string to a base64 string

string **imap_binary** (string string) \linebreak

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_body (PHP 3, PHP 4 >= 4.0.0)

Read the message body

string **imap_body** (int imap_stream, int msg_number [, int flags]) \linebreak

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_bodystruct (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Read the structure of a specified body section of a specific message

object **imap_bodystruct** (int stream_id, int msg_no, int section) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imap_check (PHP 3, PHP 4 >= 4.0.0)

Check current mailbox

object **imap_check** (int imap_stream) \linebreak

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_clearflag_full (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Clears flags on messages

string **imap_clearflag_full** (int stream, string sequence, string flag, string options) \linebreak

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_close (PHP 3, PHP 4 >= 4.0.0)

Close an IMAP stream

int **imap_close** (int imap_stream [, int flags]) \linebreak

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 >= 4.0.0)

Create a new mailbox

int **imap_createmailbox** (int imap_stream, string mbox) \linebreak

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)
    or die("can't connect: ".imap_last_error());
```

```
$name1 = "phpnewbox";
```

```

$name2 = imap_utf7_encode("phpnewb&ouml;x");

$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 >= 4.0.0)

Mark a message for deletion from current mailbox

```
int imap_delete ( int imap_stream, int msg_number [, int flags]) \linebreak
```

Returns TRUE.

imap_delete() marks messages listed in *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*.

Note: POP3 mailboxes do not have their message flags saved between connections, so *imap_expunge()* must be called during the same connection in order for messages marked for deletion to actually be purged.

Example 1. *imap_delete()* Beispiel

```
$mbox = imap_open ("{your.imap.host}INBOX", "username", "password")
    or 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 >= 4.0.0)

Delete a mailbox

```
int imap_deletemailbox ( int imap_stream, string mbox) \linebreak
```

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_errors (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

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) \linebreak

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_expunge (PHP 3, PHP 4 >= 4.0.0)

Delete all messages marked for deletion

int **imap_expunge** (int imap_stream) \linebreak

imap_expunge() deletes all the messages marked for deletion by **imap_delete()**, **imap_mail_move()**, or **imap_setflag_full()**.

Returns TRUE.

imap_fetch_overview (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Read an overview of the information in the headers of the given message

array **imap_fetch_overview** (int imap_stream, string sequence [, int flags]) \linebreak

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")
    or 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_fetchbody` (PHP 3, PHP 4 >= 4.0.0)

Fetch a particular section of the body of the message

string **imap_fetchbody** (int *imap_stream*, int *msg_number*, string *part_number* [, flags *flags*]) \linebreak

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.

See also: `imap_fetchstructure()`.

imap_fetchheader (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Returns header for a message

string **imap_fetchheader** (int imap_stream, int msgno, int flags) \linebreak

This function causes a fetch of the complete, unfiltered RFC2822 (<http://www.faqs.org/rfcs/rfc2822.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_fetchstructure (PHP 3, PHP 4 >= 4.0.0)

Read the structure of a particular message

object **imap_fetchstructure** (int imap_stream, int msg_number [, int flags]) \linebreak

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	An array of objects where each object has an "attribute" and a "value" property corresponding to the parameters on the Content-disposition MIMEheader.
ifparameters	TRUE if the parameters array exists
parameters	An array of objects where each object has an "attribute" and a "value" property.
parts	An array of objects identical in structure to the top-level object, each of which corresponds to a MIME body part.

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

See also: `imap_fetchbody()`.

`imap_get_quota` (PHP 4 >= 4.0.5)

Retrieve the quota level settings, and usage statics per mailbox

array **imap_get_quota** (int *imap_stream*, string *quota_root*) \linebreak

Returns an array with integer values *limit* and *usage* for the given mailbox. The value of *limit* represents the total amount of space allowed for this mailbox. The *usage* value represents the mailboxes current level of capacity. Will return `FALSE` in the case of failure.

This function is currently only available to users of the `c-client2000` library.

imap_stream should be the value returned from an `imap_status()` call. This stream is required to be opened as the mail admin user for the quota function to work. *quota_root* should normally be in the form of `user.name` where *name* is the mailbox you wish to retrieve information about.

Example 1. `imap_get_quota()` example

```
$mbox = imap_open("{your.imap.host}", "mailadmin", "password", OP_HALFOPEN)
    or die("can't connect: ".imap_last_error());

$quota_value = imap_get_quota($mbox, "user.kalowsky");
if(is_array($quota_value)) {
    print "Usage level is: " . $quota_value['usage'];
    print "Limit level is: " . $quota_value['limit'];
}

imap_close($mbox);
```

See also `imap_open()`, `imap_set_quota()`.

`imap_getmailboxes` (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

Read the list of mailboxes, returning detailed information on each one

array **imap_getmailboxes** (int *imap_stream*, string *ref*, string *pattern*) \linebreak

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)
    or 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_getsubscribed (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

List all the subscribed mailboxes

array **imap_getsubscribed** (int imap_stream, string ref, string pattern) \linebreak

This function is identical to `imap_getmailboxes()`, except that it only returns mailboxes that the user is subscribed to.

imap_header (PHP 3, PHP 4 >= 4.0.0)

Read the header of the message

object **imap_header** (int imap_stream, int msg_number [, int fromlength [, int subjectlength [, string default-host]]) \linebreak

This is an alias to `imap_headerinfo()` and is identical to this in any way.

imap_headerinfo (PHP 3, PHP 4 >= 4.0.0)

Read the header of the message

object **imap_headerinfo** (int imap_stream, int msg_number [, int fromlength [, int subjectlength [, string default-host]]) \linebreak

This function returns an object of various header elements.

return, date, Date, subject, Subject, in_reply_to, message_id,
newsgroups, followup_to, references

message flags:

Recent - 'R' if recent and seen,
 'N' if recent and not seen,
 ' ' if not recent
Unseen - 'U' if not seen AND not recent,
 ' ' if seen OR not seen and recent
Answered - 'A' if answered,
 ' ' if unanswered
Deleted - 'D' if deleted,
 ' ' if not deleted
Draft - 'X' if draft,
 ' ' if not draft
Flagged - 'F' if flagged,
 ' ' if not flagged

NOTE that the Recent/Unseen behavior is a little odd. If you want to know if a message is Unseen, you must check for

Unseen == 'U' || Recent == 'N'

toaddress (full to: line, up to 1024 characters)

to[] (returns an array of objects from the To line, containing):

- personal
- adl
- mailbox
- host

fromaddress (full from: line, up to 1024 characters)

from[] (returns an array of objects from the From line, containing):

- personal
- adl
- mailbox
- host

ccaddress (full cc: line, up to 1024 characters)

cc[] (returns an array of objects from the Cc line, containing):

- personal
- adl
- mailbox
- host

bccaddress (full bcc line, up to 1024 characters)

bcc[] (returns an array of objects from the Bcc line, containing):

- personal
- adl
- mailbox
- host

reply_toaddress (full reply_to: line, up to 1024 characters)

reply_to[] (returns an array of objects from the Reply_to line, containing):

- personal
- adl
- mailbox
- host

senderaddress (full sender: line, up to 1024 characters)

sender[] (returns an array of objects from the sender line, containing):

- personal
- adl

mailbox
host

return_path (full return-path: line, up to 1024 characters)
return_path[] (returns an array of objects from the return_path line,
containing):

personal
adl
mailbox
host

update (mail message date in unix time)

fetchfrom (from line formatted to fit *fromlength*
characters)

fetchsubject (subject line formatted to fit *subjectlength* characters)

imap_headers (PHP 3, PHP 4 >= 4.0.0)

Returns headers for all messages in a mailbox

array **imap_headers** (int imap_stream) \linebreak

Returns an array of string formatted with header info. One element per mail message.

imap_last_error (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

This function returns the last IMAP error (if any) that occurred during this page request

string **imap_last_error** (void) \linebreak

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_listmailbox (PHP 3, PHP 4 >= 4.0.0)

Read the list of mailboxes

array **imap_listmailbox** (int imap_stream, string ref, string pattern) \linebreak

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)
    or 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_listsubscribed (PHP 3, PHP 4 >= 4.0.0)

List all the subscribed mailboxes

array **imap_listsubscribed** (int imap_stream, string ref, string pattern) \linebreak

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_mail (PHP 3 >= 3.0.14, PHP 4 >= 4.0.0)

Send an email message

string **imap_mail** (string to, string subject, string message [, string additional_headers [, string cc [, string bcc [, string rpath]]]]) \linebreak

This function allows sending of emails with correct handling of Cc and Bcc receivers. The parameters to, cc and bcc are all strings and are all parsed as rfc822 address lists. The receivers specified in bcc will get the mail, but are excluded from the headers. Use the rpath parameter to specify return path. This is useful when using php as a mail client for multiple users.

imap_mail_compose (PHP 3 >= 3.0.5, PHP 4 >= 4.0.0)

Create a MIME message based on given envelope and body sections

string **imap_mail_compose** (array envelope, array body) \linebreak

Example 1. imap_mail_compose() example

```
<?php

$envelope["from"]="musone@afterfive.com";
$envelope["to"]="musone@darkstar";
$envelope["cc"]="musone@edgglobal.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"]=ENCBINARY;
$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_copy (PHP 3, PHP 4 >= 4.0.0)

Copy specified messages to a mailbox

int **imap_mail_copy** (int imap_stream, string msglist, string mbox [, int flags]) \linebreak

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 >= 4.0.0)

Move specified messages to a mailbox

int **imap_mail_move** (int imap_stream, string msglist, string mbox [, int flags]) \linebreak

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_mailboxmsginfo (PHP 3 >= 3.0.2, PHP 4 >= 4.0.0)

Get information about the current mailbox

object **imap_mailboxmsginfo** (int imap_stream) \linebreak

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

$mbx = imap_open("{your.imap.host}INBOX", "username", "password")
    or die("can't connect: ".imap_last_error());

$check = imap_mailboxmsginfo($mbx);

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($mbx);

?>

```

imap_mime_header_decode (PHP 3 >= 3.0.17, PHP 4 >= 4.0.0)

Decode MIME header elements

array **imap_mime_header_decode** (string text) \linebreak

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_msgno (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

This function returns the message sequence number for the given UID

int **imap_msgno** (int imap_stream, int uid) \linebreak

This function returns the message sequence number for the given UID. It is the inverse of `imap_uid()`.

imap_num_msg (PHP 3, PHP 4 >= 4.0.0)

Gives the number of messages in the current mailbox

int **imap_num_msg** (int imap_stream) \linebreak

Return the number of messages in the current mailbox.

See also: `imap_num_recent()` and `imap_status()`.

imap_num_recent (PHP 3, PHP 4 >= 4.0.0)

Gives the number of recent messages in current mailbox

int **imap_num_recent** (int imap_stream) \linebreak

Returns the number of recent messages in the current mailbox.

See also: `imap_num_msg()` and `imap_status()`.

imap_open (PHP 3, PHP 4 >= 4.0.0)

Open an IMAP stream to a mailbox

int **imap_open** (string mailbox, string username, string password [, int flags]) \linebreak

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 only 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, an optional port (prefixed by `':'`), and an optional protocol specification (prefixed by `'/'`). 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:110/pop3}INBOX", "user_id", "password");
```

To connect to an SSL IMAP or POP3 server, add /ssl after the protocol specification:

```
$mbox = imap_open ("{localhost:993/imap/ssl}INBOX", "user_id", "password");
```

To connect to an SSL IMAP or POP3 server with a self-signed certificate, add /ssl/novalidate-cert after the protocol specification:

```
$mbox = imap_open ("{localhost:995/pop3/ssl/novalidate-cert}", "user_id", "password");
```

To connect to an NNTP server on port 119 on the local server, use:

```
$nntp = imap_open ("{localhost:119/nntp}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 >= 4.0.0)

Check if the IMAP stream is still active

```
int imap_ping ( int imap_stream) \linebreak
```

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_popen (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Open a persistant IMAP stream to a mailbox

```
int imap_popen ( string mailbox, string user, string password [, int options]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

imap_qprint (PHP 3, PHP 4 >= 4.0.0)

Convert a quoted-printable string to an 8 bit string

```
string imap_qprint ( string string) \linebreak
```

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_renamemailbox (PHP 3, PHP 4 >= 4.0.0)

Rename an old mailbox to new mailbox

int **imap_renamemailbox** (int imap_stream, string old_mbox, string new_mbox) \linebreak

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 >= 4.0.0)

Reopen IMAP stream to new mailbox

int **imap_reopen** (int imap_stream, string mailbox [, string flags]) \linebreak

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 `.newsrsc` 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_rfc822_parse_adrlist (PHP 3>= 3.0.2, PHP 4 >= 4.0.0)

Parses an address string

array **imap_rfc822_parse_adrlist** (string address, string default_host) \linebreak

This function parses the address string as defined in RFC2822 (<http://www.faqs.org/rfcs/rfc2822.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>, postmaster@somedomain.net, root@somedomain.net";
$address_array = imap_rfc822_parse_adrlist($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_rfc822_parse_headers (PHP 4 >= 4.0.0)

Parse mail headers from a string

object **imap_rfc822_parse_headers** (string headers [, string defaulthost]) \linebreak

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_rfc822_write_address (PHP 3 >= 3.0.2, PHP 4 >= 4.0.0)

Returns a properly formatted email address given the mailbox, host, and personal info.

string **imap_rfc822_write_address** (string mailbox, string host, string personal) \linebreak

Returns a properly formatted email address as defined in RFC2822 (<http://www.faqs.org/rfcs/rfc2822.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_scanmailbox (PHP 3, PHP 4 >= 4.0.0)

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) \linebreak

Returns an array containing the names of the mailboxes that have *content* 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_search (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

This function returns an array of messages matching the given search criteria

array **imap_search** (int imap_stream, string criteria, int flags) \linebreak

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 uncomplete 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_set_quota (PHP 4 >= 4.0.5)

Sets a quota for a given mailbox

int **imap_set_quota** (int *imap_stream*, string *quota_root*, int *quota_limit*) \linebreak

Sets an upper limit quota on a per mailbox basis. This function requires the *imap_stream* to have been opened as the mail administrator account. It will not work if opened as any other user.

This function is currently only available to users of the c-client2000 library.

imap_stream is the stream pointer returned from a *imap_open()* call. This stream must be opened as the mail administrator, other wise this function will fail. *quota_root* is the mailbox to have a quota set. This should follow the IMAP standard format for a mailbox, 'user.name'. *quota_limit* is the maximum size (in KB) for the *quota_root*.

Returns TRUE on success and FALSE on error.

Example 1. imap_set_quota() example

```
$mbox = imap_open ("{your.imap.host:143}", "mailadmin", "password");

if(!imap_set_quota($mbox, "user.kalowsky", 3000)) {
    print "Error in setting quota\n";
    return;
}

imap_close($mbox);
```


See also `imap_open()`, `imap_set_quota()`.

imap_setacl (PHP 4 >= 4.1.0)

Sets the ACL for a given mailbox

int **imap_setacl** (int stream_id, string mailbox, string id, string rights) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imap_setflag_full (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Sets flags on messages

string **imap_setflag_full** (int stream, string sequence, string flag, string options) \linebreak

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")
    or 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_sort (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Sort an array of message headers

array **imap_sort** (int stream, int criteria, int reverse, int options) \linebreak

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_status (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

This function returns status information on a mailbox other than the current one

object **imap_status** (int imap_stream, string mailbox, int options) \linebreak

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)
    or 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_last_error()."\n";

imap_close($mbox);
```

imap_subscribe (PHP 3, PHP 4 >= 4.0.0)

Subscribe to a mailbox

int **imap_subscribe** (int imap_stream, string mbox) \linebreak

Subscribe to a new mailbox.

Returns TRUE on success and FALSE on error.

imap_thread (PHP 4 >= 4.1.0)

Return threaded by REFERENCES tree

int **imap_thread** (int stream_id [, int flags]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

imap_uid (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

This function returns the UID for the given message sequence number

int **imap_uid** (int imap_stream, int msgno) \linebreak

This function returns the UID for the given message sequence number. An UID is a 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()`.

Note: This is not supported by POP3 mailboxes.

imap_undelete (PHP 3, PHP 4 >= 4.0.0)

Unmark the message which is marked deleted

int **imap_undelete** (int imap_stream, int msg_number) \linebreak

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 >= 4.0.0)

Unsubscribe from a mailbox

int **imap_unsubscribe** (int imap_stream, string mbox) \linebreak

Unsubscribe from a specified mailbox.

Returns `TRUE` on success and `FALSE` on error.

imap_utf7_decode (PHP 3>= 3.0.15, PHP 4 >= 4.0.0)

Decodes a modified UTF-7 encoded string.

string **imap_utf7_decode** (string *text*) \linebreak

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 >= 4.0.0)

Converts 8bit data to modified UTF-7 text.

string **imap_utf7_encode** (string *data*) \linebreak

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 >= 4.0.0)

Converts text to UTF8

string **imap_utf8** (string *text*) \linebreak

Converts the given *text* to UTF8 (as defined in RFC2044 (<http://www.faqs.org/rfcs/rfc2044.html>)).

XLIV. 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 override this detection by specifying "IFX_LIBDIR", "IFX_LIBS" and "IFX_INCDIR" in the environment. The configure script will also try to detect your Informix server version. It will set the "HAVE_IFX_IUS" conditional compilation variable if your Informix version >= 9.00.

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_affected_rows (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Get number of rows affected by a query

int **ifx_affected_rows** (int *result_id*) \linebreak

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_blobinfile_mode (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Set the default blob mode for all select queries

void **ifx_blobinfile_mode** (int *mode*) \linebreak

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_byteasvarchar (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Set the default byte mode

void **ifx_byteasvarchar** (int mode) \linebreak

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_close (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Close Informix connection

int **ifx_close** ([int link_identifier]) \linebreak

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_connect (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Open Informix server connection

int **ifx_connect** ([string database [, string userid [, string password]]]) \linebreak

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_copy_blob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Duplicates the given blob object

int **ifx_copy_blob** (int bid) \linebreak

Duplicates the given blob object. *bid* is the ID of the blob object.

Returns `FALSE` on error otherwise the new blob object-id.

ifx_create_blob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Creates an blob object

int **ifx_create_blob** (int type, int mode, string param) \linebreak

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_create_char (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Creates an char object

int **ifx_create_char** (string param) \linebreak

Creates an char object. *param* should be the char content.

ifx_do (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Execute a previously prepared SQL-statement

int **ifx_do** (int result_id) \linebreak

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 >= 4.0.0)

Returns error code of last Informix call

string **ifx_error** (void) \linebreak

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 >= 4.0.0)

Returns error message of last Informix call

string **ifx_errormsg** ([int errorcode]) \linebreak

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&lt;br>", ifx_errormsg(-201));
```

ifx_fetch_row (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Get row as enumerated array

array **ifx_fetch_row** (int result_id [, mixed position]) \linebreak

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_fieldproperties (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

List of SQL fieldproperties

array **ifx_fieldproperties** (int result_id) \linebreak

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_fielddtypes (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

List of Informix SQL fields

array **ifx_fielddtypes** (int result_id) \linebreak

Returns an associative array with fieldnames as key and the SQL fieldtypes as data for query with *result_id*. Returns FALSE on error.

Example 1. Fieldnames and SQL fieldtypes

```
$types = ifx_fieldtypes ($resultid);
if (! isset ($types)) {
    ... error ...
}
for ($i = 0; $i < count($types); $i++) {
    $fname = key($types);
    printf("%s :\t type = %s\n", $fname, $types[$fname]);
    next($types);
}
```

ifx_free_blob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Deletes the blob object

```
int ifx_free_blob ( int bid) \linebreak
```

Deletes the blobobject for the given blob object-id *bid*. Returns FALSE on error otherwise TRUE.

ifx_free_char (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deletes the char object

```
int ifx_free_char ( int bid) \linebreak
```

Deletes the charobject for the given char object-id *bid*. Returns FALSE on error otherwise TRUE.

ifx_free_result (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Releases resources for the query

```
int ifx_free_result ( int result_id) \linebreak
```

Releases resources for the query associated with *result_id*. Returns FALSE on error.

ifx_get_blob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Return the content of a blob object

```
int ifx_get_blob ( int bid) \linebreak
```

Returns the content of the blob object for the given blob object-id *bid*.

ifx_get_char (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Return the content of the char object

```
int ifx_get_char ( int bid) \linebreak
```

Returns the content of the char object for the given char object-id *bid*.

ifx_getsqlca (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Get the contents of `sqlca.sqlerrd[0..5]` after a query

```
array ifx_getsqlca ( int result_id) \linebreak
```

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_htmltbl_result (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Formats all rows of a query into a HTML table

int **ifx_htmltbl_result** (int result_id [, string html_table_options]) \linebreak

Returns the number of rows fetched or FALSE on error.

Formats all rows of the *result_id* query into a html table. The optional second argument is a string of <table> tag options

Example 1. Informix results as HTML table

```
$rid = ifx_prepare ("select * from emp where name like " . $name,
                  $connid, IFX_SCROLL);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
if (! ifx_do($rid) {
    ... error ...
}

ifx_htmltbl_result ($rid, "border=\"2\"");

ifx_free_result($rid);
```

ifx_nullformat (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Sets the default return value on a fetch row

void **ifx_nullformat** (int mode) \linebreak

Sets the default return value of a NULL-value on a fetch row. Mode "0" returns "", and mode "1" returns "NULL".

ifx_num_fields (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Returns the number of columns in the query

int **ifx_num_fields** (int result_id) \linebreak

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 >= 4.0.0)

Count the rows already fetched from a query

int **ifx_num_rows** (int result_id) \linebreak

Gives the number of rows fetched so far for a query with *result_id* after a `ifx_query()` or `ifx_do()` query.

ifx_pconnect (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Open persistent Informix connection

int **ifx_pconnect** ([string database [, string userid [, string password]]) \linebreak

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_prepare (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Prepare an SQL-statement for execution

int **ifx_prepare** (string query, int conn_id [, int cursor_def, mixed blobidarray]) \linebreak

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_query (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Send Informix query

```
int ifx_query ( string query [, int link_identifier [, int cursor_type [, mixed blobidarray]]) \linebreak
```

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_textasvarchar (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Set the default text mode

```
void ifx_textasvarchar ( int mode) \linebreak
```

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_update_blob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Updates the content of the blob object

ifx_update_blob (int bid, string content) \linebreak

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_update_char (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Updates the content of the char object

int **ifx_update_char** (int bid, string content) \linebreak

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**.

ifxus_close_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Deletes the slob object

int **ifxus_close_slob** (int bid) \linebreak

Deletes the slob object on the given slob object-id *bid*. Return **FALSE** on error otherwise **TRUE**.

ifxus_create_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Creates an slob object and opens it

int **ifxus_create_slob** (int mode) \linebreak

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 >= 4.0.0)

Deletes the slob object

```
int ifxus_free_slob ( int bid) \linebreak
```

Deletes the slob object. *bid* is the Id of the slob object. Returns FALSE on error otherwise TRUE.

ifxus_open_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Opens an slob object

```
int ifxus_open_slob ( long bid, int mode) \linebreak
```

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_read_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Reads nbytes of the slob object

```
int ifxus_read_slob ( long bid, long nbytes) \linebreak
```

Reads nbytes of the slob object. *bid* is a existing slob id and *nbytes* is the number of bytes read. Return FALSE on error otherwise the string.

ifxus_seek_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Sets the current file or seek position

```
int ifxus_seek_slob ( long bid, int mode, long offset) \linebreak
```

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_tell_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Returns the current file or seek position

int **ifxus_tell_slob** (long bid) \linebreak

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_write_slob (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Writes a string into the slob object

int **ifxus_write_slob** (long bid, string content) \linebreak

Writes a string into the slob object. *bid* is a existing slob id and *content* the content to write. Return `FALSE` on error otherwise bytes written.

XLV. 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_blob_add (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Add data into created blob

```
int ibase_blob_add ( int blob_id, string data) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_cancel (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Cancel creating blob

```
int ibase_blob_cancel ( int blob_id) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_close (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Close blob

```
int ibase_blob_close ( int blob_id) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_create (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Create blob for adding data

int **ibase_blob_create** ([int link_identifier]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_echo (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Output blob contents to browser

int **ibase_blob_echo** (string blob_id_str) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_get (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Get len bytes data from open blob

string **ibase_blob_get** (int blob_id, int len) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_import (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Create blob, copy file in it, and close it

string **ibase_blob_import** ([int link_identifier, int file_id]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_info (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Return blob length and other useful info

object **ibase_blob_info** (string blob_id_str) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_blob_open (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Open blob for retrieving data parts

int **ibase_blob_open** (string blob_id) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ibase_close (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Close a connection to an InterBase database

int **ibase_close** ([int connection_id]) \linebreak

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_commit (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Commit a transaction

```
int ibase_commit ( [int link_identifier, int trans_number] ) \linebreak
```

Commits transaction *trans_number* which was created with `ibase_trans()`.

ibase_connect (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Open a connection to an InterBase database

```
int ibase_connect ( string database [, string username [, string password [, string charset [, int buffers [, int dialect  
[, string role]]]]]) \linebreak
```

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)) {
        echo $row->email, "\n";
    }
    ibase_free_result($sth);
    ibase_close($dbh);
?>
```

Note: *buffers* was added in PHP 4.0RC2.

Note: *dialect* was added in PHP 4.0RC2. It is functional only with InterBase 6 and versions higher than that.

Note: *role* was added in PHP 4.0RC2. It is functional only with InterBase 5 and versions higher than that.

See also `ibase_pconnect()`.

ibase_errmsg (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Returns error messages

```
string ibase_errmsg ( void) \linebreak
```

Returns a string containing an error message.

ibase_execute (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Execute a previously prepared query

```
int ibase_execute ( int query [, int bind_args]) \linebreak
```

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_fetch_object (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Get an object from a InterBase database

object **ibase_fetch_object** (int result_id) \linebreak

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);
?>
```

Subsequent call to **ibase_fetch_object()** would return the next row in the result set, or `FALSE` if there are no more rows.

See also `ibase_fetch_row()`.

ibase_fetch_row (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fetch a row from an InterBase database

array **ibase_fetch_row** (int result_identifier) \linebreak

Returns an array that corresponds to the fetched row, or `FALSE` if there are no more rows.

ibase_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 **ibase_fetch_row()** would return the next row in the result set, or `FALSE` if there are no more rows.

ibase_field_info (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Get information about a field

array **ibase_field_info** (int result, int field number) \linebreak

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.

```
$rs=ibase_query("SELECT * FROM tablename");
$coln = ibase_num_fields($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_query (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Free memory allocated by a prepared query

int **ibase_free_query** (int query) \linebreak

Free a query prepared by ibase_prepare().

ibase_free_result (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Free a result set

int **ibase_free_result** (int result_identifier) \linebreak

Free's a result set the has been created by ibase_query().

ibase_num_fields (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Get the number of fields in a result set

int **ibase_num_fields** (int result_id) \linebreak

Returns an integer containing the number of fields in a result set.

```
<?php
$dbh = ibase_connect ($host, $username, $password);
$stmt = 'SELECT * FROM tblname';
$stmt = ibase_query ($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()`.

ibase_pconnect (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

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]]]]]) \linebreak

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_prepare (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Prepare a query for later binding of parameter placeholders and execution

```
int ibase_prepare ( [int link_identifier, string query] ) \linebreak
```

Prepare a query for later binding of parameter placeholders and execution (via `ibase_execute()`).

ibase_query (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Execute a query on an InterBase database

```
int ibase_query ( [int link_identifier, string query [, int bind_args]] ) \linebreak
```

Performs a query on an InterBase database. If the query is not successful, returns `FALSE`. If it is successful and there are resulting rows (such as with a `SELECT` query), returns a result identifier. If the query was successful and there were no results, returns `TRUE`. Returns `FALSE` if the query fails.

See also `ibase_errmsg()`, `ibase_fetch_row()`, `ibase_fetch_object()`, and `ibase_free_result()`.

ibase_rollback (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Rolls back a transaction

```
int ibase_rollback ( [int link_identifier, int trans_number] ) \linebreak
```

Rolls back transaction *trans_number* which was created with `ibase_trans()`.

ibase_timefmt (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets the format of timestamp, date and time type columns returned from queries

```
int ibase_timefmt ( string format [, int columntype] ) \linebreak
```


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. `column_type` 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: `column_type` 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__trans (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Begin a transaction

```
int ibase__trans ( [int trans_args [, int link_identifier]]) \linebreak
```

Begins a transaction.

XLVI. Ingres II functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

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_autocommit (PHP 4 >= 4.0.2)

Switch autocommit on or off

```
bool ingres_autocommit ( [resource link]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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()`.

ingres_close (PHP 4 >= 4.0.2)

Close an Ingres II database connection

```
bool ingres_close ( [resource link]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_commit (PHP 4 >= 4.0.2)

Commit a transaction

bool **ingres_commit** ([resource link]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_connect (PHP 4 >= 4.0.2)

Open a connection to an Ingres II database

resource **ingres_connect** ([string database [, string username [, string password]]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_fetch_array` (PHP 4 >= 4.0.2)

Fetch a row of result into an array

array `ingres_fetch_array` ([int result_type [, resource link]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`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 `INGRES_NUM` for enumerated array, `INGRES_ASSOC` for associative array, or `INGRES_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_object` (PHP 4 >= 4.0.2)

Fetch a row of result into an object.

object `ingres_fetch_object` ([int *result_type* [, resource *link*]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`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: `INGRES_ASSOC`, `INGRES_NUM`, and `INGRES_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_fetch_row` (PHP 4 >= 4.0.2)

Fetch a row of result into an enumerated array

array `ingres_fetch_row` ([resource link]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`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_field_length` (PHP 4 >= 4.0.2)

Get the length of a field

`int ingres_field_length (int index [, resource link]) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`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_name` (PHP 4 >= 4.0.2)

Get the name of a field in a query result.

`string ingres_field_name (int index [, resource link]) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`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_nullable (PHP 4 >= 4.0.2)

Test if a field is nullable

```
bool ingres_field_nullable ( int index [, resource link] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_precision (PHP 4 >= 4.0.2)

Get the precision of a field

```
int ingres_field_precision ( int index [, resource link] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_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*]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_num_fields (PHP 4 >= 4.0.2)

Get the number of fields returned by the last query

int **ingres_num_fields** ([resource *link*]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_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]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_pconnect (PHP 4 >= 4.0.2)

Open a persistent connection to an Ingres II database

resource **ingres_pconnect** ([string database [, string username [, string password]]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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_query (PHP 4 >= 4.0.2)

Send a SQL query to Ingres II

bool **ingres_query** (string query [, resource link]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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 immediatly 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_rollback (PHP 4 >= 4.0.2)

Roll back a transaction

bool **ingres_rollback** ([resource link]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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()`.

XLVII. IRC Gateway Functions

What is ircg?

With ircg you can build powerful, fast and scalable webchat solutions in conjunction with dedicated or public IRC servers.

Platforms

IRCG runs under

- AIX
- FreeBSD
- HP-UX
- Irix
- Linux
- Solaris
- Tru64

Requirements

To install and use IRCG you need the following software:

1. IRCG-Library (<http://www.schumann.cx/ircg/>) from Sascha Schumann.
2. SGI Static Threads Library (<http://sourceforge.net/projects/state-threads/>)
3. thttpd (<http://www.acme.com/software/thttpd/>) webserver

Installation

A detailed installation instruction can be found here (<http://lxr.php.net/source/php4/ext/ircg/README.txt>).

ircg_channel_mode (PHP 4 >= 4.0.5)

Set channel mode flags for user

boolean **ircg_channel_mode** (resource connection, string channel, string mode_spec, string nick) \linebreak

Set channel mode flags for *channel* on server connected to by *connection*. Mode flags are passed in *mode_spec* and are applied to the user specified by *nick*.

Mode flags are set or cleared by specifying a mode character and prepending it with a plus or minus character respectively. E.g. operator mode is granted by '+o' and revoked by '-o' passed as *mode_spec*.

ircg_disconnect (PHP 4 >= 4.0.4)

Close connection to server

boolean **ircg_disconnect** (resource connection, string reason) \linebreak

ircg_disconnect() will close a *connection* to a server previously established with `ircg_pconnect()`.

See also: `ircg_pconnect()`.

ircg_fetch_error_msg (PHP 4 >= 4.1.0)

Returns the error from previous ircg operation

array **ircg_fetch_error_msg** (resource connection) \linebreak

ircg_fetch_error_msg() returns the error from the last called ircg function.

Note: Errorcode is stored in first array element, errortext in second.

Example 1. ircg_fetch_error_msg() example

```
if (!ircg_join ($id, "#php")) {
    $error = ircg_fetch_error_msg($id);
    print ("Can't join channel #php. Errorcode:
           $error[0] Description: $error[1]");
}
```

ircg_get_username (PHP 4 >= 4.1.0)

Get username for connection

string **ircg_get_username** (int connection) \linebreak

Function **ircg_get_username()** returns the username for specified connection *connection*. Returns *FALSE* if *connection* died or is not valid.

ircg_html_encode (PHP 4 >= 4.0.5)

Encodes HTML preserving output

boolean **ircg_html_encode** (string html_string) \linebreak

Encodes a HTML string *html_string* for output. This feature could be usable, e.g. if someone wants to discuss about an html problem.

ircg_ignore_add (PHP 4 >= 4.0.5)

Add a user to your ignore list on a server

boolean **ircg_ignore_add** (resource connection, string nick) \linebreak

This function will add user *nick* to your ignore list on the server connected to by *connection*. By doing so you will suppress all messages from this user from being send to you.

See also: `ircg_ignore_del()`.

ircg_ignore_del (PHP 4 >= 4.0.5)

Remove a user from your ignore list on a server

boolean **ircg_ignore_del** (resource connection, string nick) \linebreak

This function remove user *nick* from your ignore list on the server connected to by *connection*.

See also: `ircg_ignore_add()`.

ircg_is_conn_alive (PHP 4 >= 4.0.5)

Check connection status

boolean **ircg_is_conn_alive** (resource connection) \linebreak

ircg_is_conn_alive() returns `TRUE` if *connection* is still alive and working or `FALSE` if the server no longer talks to us.

ircg_join (PHP 4 >= 4.0.4)

Join a channel on a connected server

boolean **ircg_join** (resource connection, string channel) \linebreak

Join the channel *channel* on the server connected to by *connection*.

ircg_kick (PHP 4 >= 4.0.5)

Kick a user out of a channel on server

boolean **ircg_kick** (resource connection, string channel, string nick, string reason) \linebreak

Kick user *nick* from *channel* on server connected to by *connection*. *reason* should give a short message describing why this action was performed.

ircg_lookup_format_messages (PHP 4 >= 4.0.5)

Select a set of format strings for display of IRC messages

boolean **ircg_lookup_format_messages** (string name) \linebreak

Select the set of format strings to use for display of IRC messages and events. Sets may be registered with `ircg_register_format_messages()`, a default set named `ircg` is always available.

See also: `ircg_register_format_messages()`

ircg_msg (PHP 4 >= 4.0.4)

Send message to channel or user on server

boolean **ircg_msg** (resource connection, string recipient, string message [, boolean suppress]) \linebreak

ircg_msg() will send the message to a channel or user on the server connected to by *connection*. A *recipient* starting with # or & will send the *message* to a channel, anything else will be interpreted as a username.

Setting the optional parameter *suppress* to a `TRUE` value will suppress output of your message to your own *connection*.

ircg_nick (PHP 4 >= 4.0.5)

Change nickname on server

boolean **ircg_nick** (resource connection, string nick) \linebreak

Change your nickname on the given *connection* to the one given in *nick* if possible.

Will return `TRUE` on success and `FALSE` on failure.

ircg_nickname_escape (PHP 4 >= 4.0.6)

Encode special characters in nickname to be IRC-compliant

string **ircg_nickname_escape** (string nick) \linebreak

Function **ircg_nickname_escape()** returns a decoded nickname specified by *nick* wich is IRC-compliant.

See also: `ircg_nickname_unescape()`

ircg_nickname_unescape (PHP 4 >= 4.0.6)

Decodes encoded nickname

string **ircg_nickname_unescape** (string nick) \linebreak

Function **ircg_nickname_unescape()** returns a decoded nickname, which is specified in *nick*.

See also: `ircg_nickname_escape()`

ircg_notice (PHP 4 >= 4.0.5)

Send a notice to a user on server

boolean **ircg_notice** (resource connection, string , string message) \linebreak

This function will send the *message* text to the user *nick* on the server connected to by *connection*. Query your IRC documentation of choice for the exact difference between a `MSG` and a `NOTICE`.

ircg_part (PHP 4 >= 4.0.4)

Leave a channel on server

boolean **ircg_part** (resource connection, string channel) \linebreak

Leave the channel *channel* on the server connected to by *connection*.

ircg_pconnect (PHP 4 >= 4.0.4)

Connect to an IRC server

resource **ircg_pconnect** (string username [, string server_ip [, int server_port [, string msg_format [, array ctcp_messages [, array user_settings]]]]) \linebreak

ircg_pconnect() will try to establish a connection to an IRC server and return a connection resource handle for further use.

The only mandatory parameter is *username*, this will set your initial nickname on the server. *server_ip* and *server_port* are optional and default to 127.0.0.1 and 6667.

Note: For now parameter *server_ip* will not do any hostname lookups and will only accept IP addresses in numerical form.

Currently **ircg_pconnect()** always returns a valid handle, even if the connection failed.

You can customize the output of IRC messages and events by selection a format string set previously created with `ircg_register_format_messages()` by specifying the sets name in *msg_format*.

ctcp_messages

user_settings

See also: `ircg_disconnect()`, `ircg_is_conn_alive()`, `ircg_register_format_messages()`.

ircg_register_format_messages (PHP 4 >= 4.0.5)

Register a set of format strings for display of IRC messages

boolean **ircg_register_format_messages** (string name, array messages) \linebreak

With **ircg_register_format_messages()** you can customize the way your IRC output looks like. You can even register different format string sets and switch between them on the fly with `ircg_lookup_format_messages()`.

- Plain channel message
- Private message received
- Private message sent
- Some user leaves channel
- Some user enters channel
- Some user was kicked from the channel
- Topic has been changed
- Error
- Fatal error
- Join list end(?)
- Self part(?)
- Some user changes his nick
- Some user quits his connection
- Mass join begin
- Mass join element
- Mass join end
- Whois user
- Whois server
- Whois idle
- Whois channel
- Whois end
- Voice status change on user
- Operator status change on user
- Banlist
- Banlist end

- %f - from
- %t - to
- %c - channel
- %r - plain message
- %m - encoded message
- %j - js encoded message

- 1 - mod encode
- 2 - nickname decode

See also: `ircg_lookup_format_messages()`.

ircg_set_current (PHP 4 >= 4.0.4)

Set current connection for output

boolean **ircg_set_current** (resource connection) \linebreak

Select the current connection for output in this execution context. Every output sent from the server connected to by *connection* will be copied to standard output while using default formatting or a format string set specified by `ircg_register_format_messages()` and selected by `ircg_lookup_format_messages()`.

See also: `ircg_register_format_messages()` and `ircg_lookup_format_messages()`.

ircg_set_file (PHP 4 >= 4.2.0)

Set logfile for connection

bool **ircg_set_file** (int connection, string path) \linebreak

Function **ircg_set_file()** specifies a logfile *path* in which all output from connection *connection* will be logged. Returns `TRUE` on success, otherwise `FALSE`.

ircg_set_on_die (PHP 4 >= 4.2.0)

Set hostaction to be execute when connection dies

bool **ircg_set_on_die** (int connection, string host, int port, string data) \linebreak

In case of the termination of connection *connection* IRCG will connect to *host* at *port* (Note: host must be an IPv4 address, IRCG does not resolve host-names due to blocking issues), send *data* to the new host connection and will wait until the remote part closes connection. This can be used to trigger a php script for example.

ircg_topic (PHP 4 >= 4.0.5)

Set topic for channel on server

boolean **ircg_topic** (resource connection, string channel, string new_topic) \linebreak

Change the topic for channel *channel* on the server connected to by *connection* to *new_topic*.

ircg_whois (PHP 4 >= 4.0.5)

Query user information for nick on server

boolean **ircg_whois** (resource connection, string nick) \linebreak

Sends a query to the connected server *connection* to send information for the specified user *nick*.

XLVIII. Java

There are two possible ways to bridge PHP and Java: you can either integrate PHP into a Java Servlet environment, which is the more stable and efficient solution, or integrate Java support into PHP. The former is provided by a SAPI module that interfaces with the Servlet server, the latter by the Java extension.

PHP 4 ext/java provides a simple and effective means for creating and invoking methods on Java objects from PHP. The JVM is created using JNI, and everything runs in-process. Build instructions for ext/java can be found in php4/ext/java/README.

Example 1. Java Example

```
<?php
// get instance of Java class java.lang.System in PHP
$system = new Java('java.lang.System');

// demonstrate property access
print 'Java version=' . $system->getProperty('java.version') . ' <br>';
print 'Java vendor=' . $system->getProperty('java.vendor') . ' <br>';
print 'OS=' . $system->getProperty('os.name') . ' ' .
      $system->getProperty('os.version') . ' on ' .
      $system->getProperty('os.arch') . ' <br>';

// java.util.Date example
$formatter = new Java('java.text.SimpleDateFormat',
                      "EEEE, MMMM dd, yyyy 'at' h:mm:ss a zzzz");

print $formatter->format(new Java('java.util.Date'));
?>
```

Example 2. AWT Example

```
<?php
// This example is only intended to be run as a CGI.

$frame = new Java('java.awt.Frame', 'PHP');
$button = new Java('java.awt.Button', 'Hello Java World!');

$frame->add('North', $button);
$frame->validate();
$frame->pack();
$frame->visible = True;

$thread = new Java('java.lang.Thread');
$thread->sleep(10000);
```

```
$frame->dispose();
?>
```

Notes:

- `new Java()` will create an instance of a class if a suitable constructor is available. If no parameters are passed and the default constructor is useful as it provides access to classes like `java.lang.System` which expose most of their functionality through static methods.
- Accessing a member of an instance will first look for bean properties then public fields. In other words, `print $date.time` will first attempt to be resolved as `$date.getTime()`, then as `$date.time`.
- Both static and instance members can be accessed on an object with the same syntax. Furthermore, if the java object is of type `java.lang.Class`, then static members of the class (fields and methods) can be accessed.
- Exceptions raised result in PHP warnings, and `NULL` results. The warnings may be eliminated by prefixing the method call with an "@" sign. The following APIs may be used to retrieve and reset the last error:
 - `java_last_exception_get()`
 - `java_last_exception_clear()`

- Overload resolution is in general a hard problem given the differences in types between the two languages. The PHP Java extension employs a simple, but fairly effective, metric for determining which overload is the best match.

Additionally, method names in PHP are not case sensitive, potentially increasing the number of overloads to select from.

Once a method is selected, the parameters are coerced if necessary, possibly with a loss of data (example: double precision floating point numbers will be converted to boolean).

- In the tradition of PHP, arrays and hashtables may pretty much be used interchangeably. Note that hashtables in PHP may only be indexed by integers or strings; and that arrays of primitive types in Java can not be sparse. Also note that these constructs are passed by value, so may be expensive in terms of memory and time.

`sapi/servlet` builds upon the mechanism defined by `ext/java` to enable the entire PHP processor to be run as a servlet. The primary advantage of this from a PHP perspective is that web servers which support servlets typically take great care in pooling and reusing JVMs. Build instructions for the Servlet SAPI module can be found in `php4/sapi/README`. Notes:

- While this code is intended to be able to run on any servlet engine, it has only been tested on Apache's Jakarta/tomcat to date. Bug reports, success stories and/or patches required to get this code

to run on other engines would be appreciated.

- PHP has a habit of changing the working directory. sapi/servlet will eventually change it back, but while PHP is running the servlet engine may not be able to load any classes from the CLASSPATH which are specified using a relative directory syntax, or find the work directory used for administration and JSP compilation tasks.

java_last_exception_clear (PHP 4 >= 4.0.2)

Clear last Java exception

```
void java_last_exception_clear ( void) \linebreak
```

java_last_exception_get (PHP 4 >= 4.0.2)

Get last Java exception

```
exception java_last_exception_get ( void) \linebreak
```

The following example demonstrates the usage of Java's exception handler from within PHP:

Example 1. Java exception handler

```
<?php
    $stack = new Java('java.util.Stack');
    $stack->push(1);

    // This should succeed
    $result = $stack->pop();
    $ex = java_last_exception_get();
    if (!$ex) print "$result\n";

    // This should fail (error suppressed by @)
    $result = @$stack->pop();
    $ex = java_last_exception_get();
    if ($ex) print $ex->toString();

    // Clear last exception
    java_last_exception_clear();
?>
```

XLIX. 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 entires returned is ".ldap_count_entries($ds,$sr)."<p>";

    echo "Getting entries ...<p>";
    $info = ldap_get_entries($ds, $sr);
    echo "Data for ".$info["count"]." items returned:<p>";

    for ($i=0; $i<$info["count"]; $i++) {
        echo "dn is: ". $info[$i]["dn"] ."<br>";
        echo "first cn entry is: ". $info[$i]["cn"][0] ."<br>";
        echo "first email entry is: ". $info[$i]["mail"][0] ."<p>";
    }

    echo "Closing connection";
    ldap_close($ds);
} else {
    echo "<h4>Unable to connect to LDAP server</h4>";
}
?>

```

Using the PHP LDAP calls

You will need to get and compile LDAP client libraries from either the University of Michigan ldap-3.3 package or the Netscape Directory SDK 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://www.innosoft.com/ldapworld>)

The Netscape SDK contains a helpful Programmer's Guide in .html format.

ldap_8859_to_t61 (PHP 4 >= 4.0.2)

Translate 8859 characters to t61 characters

string **ldap_8859_to_t61** (string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_add (PHP 3, PHP 4 >= 4.0.0)

Add entries to LDAP directory

bool **ldap_add** (resource link_identifier, string dn, array entry) \linebreak

Returns `TRUE` on success, `FALSE` on failure.

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 >= 4.0.0)

Bind to LDAP directory

```
bool ldap_bind ( resource link_identifier [, string bind_rdn [, string bind_password]]) \linebreak
```

Binds to the LDAP directory with specified RDN and password. Returns `TRUE` on success, `FALSE` on failure.

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 >= 4.0.0)

Close link to LDAP server

```
bool ldap_close ( resource link_identifier) \linebreak
```

Returns `TRUE` on success, `FALSE` on failure.

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()**.

Note: This function is an alias of `ldap_unbind()`.

ldap_compare (PHP 4 >= 4.0.2)

Compare value of attribute found in entry specified with DN

```
bool ldap_compare ( resource link_identifier, string dn, string attribute, string value) \linebreak
```

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.";
}
?>
```

Warning

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 >= 4.0.0)

Connect to an LDAP server

resource **ldap_connect** ([string hostname [, int port]]) \linebreak

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.

Note: URL and SSL support were added in 4.0.4.

ldap_count_entries (PHP 3, PHP 4 >= 4.0.0)

Count the number of entries in a search

int **ldap_count_entries** (resource link_identifier, resource result_identifier) \linebreak

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 >= 4.0.0)

Delete an entry from a directory

bool **ldap_delete** (resource link_identifier, string dn) \linebreak

Returns `TRUE` on success, `FALSE` on failure.

ldap_delete() function delete a particular entry in LDAP directory specified by *dn*.

ldap_dn2ufn (PHP 3, PHP 4 >= 4.0.0)

Convert DN to User Friendly Naming format

string **ldap_dn2ufn** (string *dn*) \linebreak

ldap_dn2ufn() function is used to turn a DN, specified by *dn*, into a more user-friendly form, stripping off type names.

ldap_err2str (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Convert LDAP error number into string error message

string **ldap_err2str** (int *errno*) \linebreak

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 >= 4.0.0)

Return the LDAP error number of the last LDAP command

int **ldap_errno** (resource *link_identifier*) \linebreak

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 `php.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 >= 4.0.0)

Return the LDAP error message of the last LDAP command

string **ldap_error** (resource *link_identifier*) \linebreak

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 `php.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 >= 4.0.0)

Splits DN into its component parts

array **ldap_explode_dn** (string dn, int with_attrib) \linebreak

ldap_explode_dn() function is used to split the 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 >= 4.0.0)

Return first attribute

string **ldap_first_attribute** (resource link_identifier, resource result_entry_identifier, int ber_identifier) \linebreak

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 *result_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 >= 4.0.0)

Return first result id

resource **ldap_first_entry** (resource link_identifier, resource result_identifier) \linebreak

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_first_reference (PHP 4 >= 4.0.5)

Return first reference

resource **ldap_first_reference** (resource link, resource result) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_free_result (PHP 3, PHP 4 >= 4.0.0)

Free result memory

bool **ldap_free_result** (resource result_identifier) \linebreak

Returns TRUE on success, FALSE on failure.

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 >= 4.0.0)

Get attributes from a search result entry

array **ldap_get_attributes** (resource link_identifier, resource result_entry_identifier) \linebreak

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] = (i+1)th 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 >= 4.0.0)

Get the DN of a result entry

```
string ldap_get_dn ( resource link_identifier, resource result_entry_identifier) \linebreak
```

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 >= 4.0.0)

Get all result entries

```
array ldap_get_entries ( resource link_identifier, resource result_identifier) \linebreak
```

Returns a complete result information in a multi-dimensional array on success and `FALSE` on error.

`ldap_get_entries()` function is used to simplify reading multiple entries from the result, specified with *result_identifier*, and then reading the attributes and multiple values. The entire information is returned by one function call in a multi-dimensional array. The structure of the array is as follows.

The attribute index is converted to lowercase. (Attributes are case-insensitive for directory servers, but not when used as array indices.)

`return_value["count"]` = number of entries in the result
`return_value[0]` : refers to the details of first entry

`return_value[i]["dn"]` = DN of the *i*th entry in the result

`return_value[i]["count"]` = number of attributes in *i*th entry
`return_value[i][j]` = *j*th attribute in the *i*th entry in the result

`return_value[i]["attribute"]["count"]` = number of values for
attribute in *i*th entry
`return_value[i]["attribute"][j]` = *j*th value of attribute in *i*th entry

See also `ldap_first_entry()` and `ldap_next_entry()`

ldap_get_option (PHP 4 >= 4.0.4)

Get the current value for given option

`bool ldap_get_option (resource link_identifier, int option, mixed retval) \linebreak`

Sets *retval* to the value of the specified option. Returns `TRUE` on success, `FALSE` on failure.

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 `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>)

Note: 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 >= 4.0.0)

Get all values from a result entry

array **ldap_get_values** (resource link_identifier, resource result_entry_identifier, string attribute) \linebreak

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 >= 4.0.0)

Get all binary values from a result entry

array **ldap_get_values_len** (resource link_identifier, resource result_entry_identifier, string attribute) \linebreak

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 >= 4.0.0)

Single-level search

resource **ldap_list** (resource link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]) \linebreak

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] ;

```

Note: From 4.0.5 on it's also possible to do parallel searches. See `ldap_search()` for details.

ldap_mod_add (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Add attribute values to current attributes

`bool ldap_mod_add (resource link_identifier, string dn, array entry) \linebreak`

Returns `TRUE` on success, `FALSE` on failure.

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 >= 4.0.0)

Delete attribute values from current attributes

`bool ldap_mod_del (resource link_identifier, string dn, array entry) \linebreak`

Returns `TRUE` on success, `FALSE` on failure.

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_delete()` function.

ldap_mod_replace (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Replace attribute values with new ones

`bool ldap_mod_replace (resoure link_identifier, string dn, array entry) \linebreak`

Returns `TRUE` on success, `FALSE` on failure.

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_modify (PHP 3, PHP 4 >= 4.0.0)

Modify an LDAP entry

bool **ldap_modify** (resource link_identifier, string dn, array entry) \linebreak

Returns TRUE on success, FALSE on failure.

ldap_modify() function is used to modify the existing entries in the LDAP directory. The structure of the entry is same as in `ldap_add()`.

ldap_next_attribute (PHP 3, PHP 4 >= 4.0.0)

Get the next attribute in result

string **ldap_next_attribute** (resource link_identifier, resource result_entry_identifier, resource ber_identifier) \linebreak

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 >= 4.0.0)

Get next result entry

resource **ldap_next_entry** (resource link_identifier, resource result_entry_identifier) \linebreak

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_entry_identifier* as returned from the `ldap_first_entry()`.

See also `ldap_get_entries()`

ldap_next_reference (PHP 4 >= 4.0.5)

Get next reference

resource **ldap_next_reference** (resource link, resource entry) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_parse_reference (PHP 4 >= 4.0.5)

Extract information from reference entry

bool **ldap_parse_reference** (resource link, resource entry, array referrals) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_parse_result (PHP 4 >= 4.0.5)

Extract information from result

bool **ldap_parse_result** (resource link, resource result, int errcode, string matcheddn, string errmsg, array referrals) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_read (PHP 3, PHP 4 >= 4.0.0)

Read an entry

resource **ldap_read** (resource link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]) \linebreak

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*.

From 4.0.5 on it's also possible to do parallel searches. See ldap_search() for details.

ldap_rename (PHP 4 >= 4.0.5)

Modify the name of an entry

bool **ldap_rename** (resource link_identifier, string dn, string newrdn, string newparent, bool deleteoldrdn) \linebreak

The entry specified by *dn* is renamed/moved. The new RDN is specified by *newrdn* and the new parent/superior entry is specified by *newparent*. If the parameter *deleteoldrdn* is TRUE the old RDN value(s) is removed, else the old RDN value(s) is retained as non-distinguished values of the entry. Returns TRUE on success, FALSE on failure.

Note: This function currently only works with LDAPv3. You may have to use ldap_set_option() prior to binding to use LDAPv3. This function is only available when using OpenLDAP 2.x.x OR Netscape Directory SDK x.x, and was added in PHP 4.0.5.

ldap_search (PHP 3, PHP 4 >= 4.0.0)

Search LDAP tree

resource **ldap_search** (resource link_identifier, string base_dn, string filter [, array attributes [, int attrsonly [, int sizelimit [, int timelimit [, int deref]]]]) \linebreak

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 an 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.

Note: 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 doumentation (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>";

```

From 4.0.5 on it's also possible to do parallel searches. To do this you use an array of link identifiers, rather than a single identifier, as the first argument. If you don't want the same base DN and the same filter for all the searches, you can also use an array of base DNs and/or an array of filters. Those arrays must be of the same size as the link identifier array since the first entries of the arrays are used for one search, the second entries are used for another, and so on. When doing parallel searches an array of search result identifiers is returned, except in case of error, then the entry corresponding to the search will be `FALSE`. This is very much like the value normally returned, except that a result identifier is always returned when a search was made. There are some rare cases where the normal search returns `FALSE` while the parallel search returns an identifier.

ldap_set_option (PHP 4 >= 4.0.4)

Set the value of the given option

`bool ldap_set_option (resource link_identifier, int option, mixed newval) \linebreak`

Sets the value of the specified option to be *newval*. Returns `TRUE` on success, `FALSE` on failure. 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.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.

Note: 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_set_rebind_proc (PHP 4 >= 4.2.0)

Set a callback function to do re-binds on referral chasing.

```
bool ldap_set_rebind_proc ( resource link, string callback ) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ldap_sort (PHP 4 >= 4.2.0)

Sort LDAP result entries

bool **ldap_sort** (resource link, resource result, string sortfilter) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_start_tls (PHP 4 >= 4.2.0)

Start TLS

bool **ldap_start_tls** (resource link) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_t61_to_8859 (PHP 4 >= 4.0.2)

Translate t61 characters to 8859 characters

string **ldap_t61_to_8859** (string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

ldap_unbind (PHP 3, PHP 4 >= 4.0.0)

Unbind from LDAP directory

bool **ldap_unbind** (resource link_identifier) \linebreak

Returns TRUE on success, FALSE on failure.

ldap_unbind() function unbinds from the LDAP directory.

L. Mail functions

The mail() function allows you to send mail

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`.

ezmlm_hash (PHP 3 >= 3.0.17, PHP 4 >= 4.0.2)

Calculate the hash value needed by EZMLM

```
int ezmlm_hash ( string addr) \linebreak
```

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 = "joecool@example.com";
$hash = ezmlm_hash ($user);
$query = sprintf ("INSERT INTO sample VALUES (%s, '%s')", $hash, $user);
$db->query($query); // using PHP LIB db interface
```

mail (PHP 3, PHP 4 >= 4.0.0)

send mail

```
bool mail ( string to, string subject, string message [, string additional_headers [, string additional_parameters]])
\linebreak
```

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*. Email with attachments and special types of content can be sent using this function. This is accomplished via MIME-encoding - for more information, see this [Zend article \(http://www.zend.com/zend/spotlight/sendmimeemailpart1.php\)](http://www.zend.com/zend/spotlight/sendmimeemailpart1.php) or the [PEAR Mime Classes \(http://pear.php.net/get/Mail_Mime\)](http://pear.php.net/get/Mail_Mime).

The following RFC's may also be useful: RFC 1896 (<http://www.ietf.org/rfc/rfc1896.txt>), RFC 2045 (<http://www.ietf.org/rfc/rfc2045.txt>), RFC 2046 (<http://www.ietf.org/rfc/rfc2046.txt>), RFC 2047 (<http://www.ietf.org/rfc/rfc2047.txt>), RFC 2048 (<http://www.ietf.org/rfc/rfc2048.txt>), and RFC 2049 (<http://www.ietf.org/rfc/rfc2049.txt>).

mail() returns `TRUE` if the mail is successfully sent, `FALSE` otherwise.

Example 1. Sending mail.

```
mail("joecool@example.com", "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 carriage return and newline.

Note: You must use `\r\n` to separate headers, although some Unix mail transfer agents may work with just a single newline (`\n`). The Cc: header is case sensitive and must be written as `cc:` on Win32 systems. The Bcc: header is also not supported on Win32 systems.

Example 2. Sending mail with extra headers.

```
mail("nobody@example.com", "the subject", $message,
     "From: webmaster@$SERVER_NAME\r\n"
     ."Reply-To: webmaster@$SERVER_NAME\r\n"
     ."X-Mailer: PHP/" . phpversion());
```

The `additional_parameters` parameter can be used to pass additional parameters to the program configured to use when sending mail using the `sendmail_path` configuration setting. For example, this can be used to set the envelope sender address when using `sendmail`. You may need to add the user that your web server runs as to your `sendmail` configuration to prevent a 'X-Warning' header from being added to the message when you set the envelope sender using this method.

Example 3. Sending mail with extra headers and setting an additional command line parameter.

```
mail("nobody@example.com", "the subject", $message,
     "From: webmaster@$SERVER_NAME", "-fwebmaster@$SERVER_NAME");
```

Note: This fifth parameter was added in PHP 4.0.5.

You can also use simple string building techniques to build complex email messages.

Example 4. Sending complex email.

```
/* recipients */
$to = "Mary <mary@example.com>" . ", " ; //note the comma
$to .= "Kelly <kelly@example.com>";

/* subject */
```

```

$subject = "Birthday Reminders for August";

/* message */
$message = '
<html>
<head>
  <title>Birthday Reminders for August</title>
</head>
<body>
<p>Here are the birthdays upcoming in August!</p>
<table>
  <tr>
    <th>Person</th><th>Day</th><th>Month</th><th>Year</th>
  </tr>
  <tr>
    <td>Joe</td><td>3rd</td><td>August</td><td>1970</td>
    <td>Sally</td><td>17th</td><td>August</td><td>1973</td>
  </tr>
</table>
</body>
</html>
';

/* To send HTML mail, you can set the Content-type header. */
$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-type: text/html; charset=iso-8859-1\r\n";

/* additional headers */
$headers .= "From: Birthday Reminder <birthday@example.com>\r\n";

$headers .= "Cc: birthdayarchive@example.com\r\n";
$headers .= "Bcc: birthdaycheck@example.com\r\n";

/* and now mail it */
mail($to, $subject, $message, $headers);

```

Note: Make sure you do not have any newline characters in the *to* or *subject*, or the mail may not be sent properly.

LI. mailparse functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

mailparse_determine_best_xfer_encoding (4.1.0 - 4.2.0 only)

Figures out the best way of encoding the content read from the file pointer fp, which must be seek-able

```
int mailparse_determine_best_xfer_encoding ( resource fp) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_create (4.1.0 - 4.2.0 only)

Returns a handle that can be used to parse a message

```
int mailparse_msg_create ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_extract_part (4.1.0 - 4.2.0 only)

Extracts/decodes a message section. If callbackfunc is not specified, the contents will be sent to "stdout"

```
void mailparse_msg_extract_part ( resource rfc2045, string msgbody [, string callbackfunc]) \linebreak
```


Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_extract_part_file (4.1.0 - 4.2.0 only)

Extracts/decodes a message section, decoding the transfer encoding

string **mailparse_msg_extract_part_file** (resource rfc2045, string filename [, string callbackfunc]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_free (4.1.0 - 4.2.0 only)

Frees a handle allocated by mailparse_msg_crea

void **mailparse_msg_free** (resource rfc2045buf) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_get_part (4.1.0 - 4.2.0 only)

Returns a handle on a given section in a mimemessage

int **mailparse_msg_get_part** (resource rfc2045, string mimesection) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_get_part_data (4.1.0 - 4.2.0 only)

Returns an associative array of info about the message

array **mailparse_msg_get_part_data** (resource rfc2045) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_get_structure (4.1.0 - 4.2.0 only)

Returns an array of mime section names in the supplied message

array **mailparse_msg_get_structure** (resource rfc2045) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_parse (4.1.0 - 4.2.0 only)

Incrementally parse data into buffer

void **mailparse_msg_parse** (resource rfc2045buf, string data) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_msg_parse_file (4.1.0 - 4.2.0 only)

Parse file and return a resource representing the structure

resource **mailparse_msg_parse_file** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_rfc822_parse_addresses (4.1.0 - 4.2.0 only)

Parse addresses and returns a hash containing that data

array **mailparse_rfc822_parse_addresses** (string addresses) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_stream_encode (4.1.0 - 4.2.0 only)

Streams data from source file pointer, apply encoding and write to destfp

bool **mailparse_stream_encode** (resource sourcefp, resource destfp, string encoding) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

mailparse_uudecode_all (PHP 4.4.2.0 only)

Scans the data from fp and extract each embedded uuencoded file. Returns an array listing filename information

array **mailparse_uudecode_all** (resource fp) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

LII. Mathematical Functions

Introduction

These math functions will only handle values within the range of the integer and float types on your computer. (this corresponds currently to the C types long resp. double) 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 PHP 4.0.0. All other constants are available starting with PHP 4.0.0. Constants labeled [4.0.2] were added in PHP 4.0.2.

abs (PHP 3, PHP 4 >= 4.0.0)

Absolute value

mixed **abs** (mixed number) \linebreak

Returns the absolute value of number. If the argument number is of type float, the return type is also float, otherwise it is integer (as float usually has a bigger value range than integer).

Example 1. abs() example

```
$abs = abs(-4.2); // $abs = 4.2; (double/float)
$abs2 = abs(5); // $abs2 = 5; (integer)
$abs3 = abs(-5); // $abs3 = 5; (integer)
```

acos (PHP 3, PHP 4 >= 4.0.0)

Arc cosine

float **acos** (float arg) \linebreak

Returns the arc cosine of *arg* in radians.

See also acosh(), asin() and atan().

acosh (PHP 4 >= 4.1.0)

Inverse hyperbolic cosine

float **acosh** (float arg) \linebreak

Returns the inverse hyperbolic cosine of *arg*, i.e. the value whose hyperbolic cosine is *arg*.

Note: This function is not implemented on Windows platforms.

See also acos(), asin() and atan().

asin (PHP 3, PHP 4 >= 4.0.0)

Arc sine

float **asin** (float *arg*) \linebreak

Returns the arc sine of *arg* in radians.

See also asinh(), acos() and atan().

asinh (PHP 4 >= 4.1.0)

Inverse hyperbolic sine

float **asinh** (float *arg*) \linebreak

Returns the inverse hyperbolic sine of *arg*, i.e. the value whose hyperbolic sine is *arg*.

Note: This function is not implemented on Windows platforms.

See also asin(), acos() and atan().

atan (PHP 3, PHP 4 >= 4.0.0)

Arc tangent

float **atan** (float *arg*) \linebreak

Returns the arc tangent of *arg* in radians.

See also atanh(), asin() and acos().

atan2 (PHP 3 >= 3.0.5, PHP 4 >= 4.0.0)

arc tangent of two variables

float **atan2** (float *y*, float *x*) \linebreak

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 π (inclusive).

See also acos() and atan().

atanh (PHP 4 >= 4.1.0)

Inverse hyperbolic tangent

float **atanh** (float *arg*) \linebreak

Returns the inverse hyperbolic tangent of *arg*, i.e. the value whose hyperbolic tangent is *arg*.

Note: This function is not implemented on Windows platforms.

See also atan(), asin() and acos().

base_convert (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Convert a number between arbitrary bases

string **base_convert** (string *number*, int *frombase*, int *tobase*) \linebreak

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 >= 4.0.0)

Binary to decimal

int **bindec** (string *binary_string*) \linebreak

Returns the decimal equivalent of the binary number represented by the *binary_string* argument.

bindec() converts a binary number to an integer. 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 >= 4.0.0)

Round fractions up

float **ceil** (float value) \linebreak

Returns the next highest integer value by rounding up *value* if necessary. The return value of **ceil()** is still of type float as the value range of float is usually bigger than that of int.

Example 1. ceil() examples

```
$five = ceil(4.3); // $five = 5.0;  
$ten = ceil(9.999); // $ten = 10.0;
```

See also floor() and round().

cos (PHP 3, PHP 4 >= 4.0.0)

Cosine

float **cos** (float arg) \linebreak

Returns the cosine of arg in radians.

See also sin() and tan().

cosh (PHP 4 >= 4.1.0)

Hyperbolic cosine

float **cosh** (float arg) \linebreak

Returns the hyperbolic cosine of *arg*, defined as $(\exp(\text{arg}) + \exp(-\text{arg})) / 2$.

See also cos(), acosh(), sin() and tan().

decbin (PHP 3, PHP 4 >= 4.0.0)

Decimal to binary

string **decbin** (int number) \linebreak

Returns a string containing a binary representation of the given number argument. The largest number that can be converted is 4294967295 in decimal resulting to a string of 32 1's.

See also the `bindec()` function.

dechex (PHP 3, PHP 4 >= 4.0.0)

Decimal to hexadecimal

string **dechex** (int number) \linebreak

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 `hexdec()`.

decoct (PHP 3, PHP 4 >= 4.0.0)

Decimal to octal

string **decoct** (int number) \linebreak

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 "1777777777".

See also `octdec()`.

deg2rad (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Converts the number in degrees to the radian equivalent

float **deg2rad** (float number) \linebreak

This function converts *number* from degrees to the radian equivalent.

See also `rad2deg()`.

exp (PHP 3, PHP 4 >= 4.0.0)

e to the power of ...

float **exp** (float arg) \linebreak

Returns e raised to the power of *arg*.

See also `pow()`.

expm1 (PHP 4 >= 4.1.0)

Returns $\exp(\text{number}) - 1$, computed in a way that accurate even when the value of number is close to zero

float **expm1** (float number) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

floor (PHP 3, PHP 4 >= 4.0.0)

Round fractions down

float **floor** (float value) \linebreak

Returns the next lowest integer value by rounding down *value* if necessary. The return value of **floor()** is still of type float because the value range of float is usually bigger than that of int.

Example 1. floor() examples

```
$four = floor(4.3); // $four = 4.0;
$nine = floor(9.999); // $nine = 9.0;
```

See also `ceil()` and `round()`.

getrandmax (PHP 3, PHP 4 >= 4.0.0)

Show largest possible random value

```
int getrandmax ( void) \linebreak
```

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 >= 4.0.0)

Hexadecimal to decimal

```
int hexdec ( string hex_string) \linebreak
```

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.

hexdec() will replace of any non-hexadecimal characters it encounters by 0. This way, all left zeros are ignored, but right zeros will be valued.

Example 1. hexdec() example

```
var_dump(hexdec("See"));
var_dump(hexdec("ee"));
// both prints "int(238)"

var_dump(hexdec("that"));
var_dump(hexdec("a0"));
// both prints int(160)
```

See also dechex().

hypot (PHP 4 >= 4.1.0)

Returns $\sqrt{\text{num1}^2 + \text{num2}^2}$

```
float hypot ( float num1, float num2) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

is_finite (PHP 4 >= 4.2.0)

bool **is_finite** (float *val*) \linebreak

Returns `TRUE` if *val* is a legal finite number within the allowed range for a PHP float on this platform.

is_infinite (PHP 4 >= 4.2.0)

bool **is_infinite** (float *val*) \linebreak

Returns `TRUE` if *val* is infinite (positive or negative), like the result of `log(0)` or any value too big to fit into a float on this platform.

is_nan (PHP 4 >= 4.2.0)

bool **is_nan** (float *val*) \linebreak

Returns `TRUE` if *val* is 'not a number', like the result of `acos(1.01)`.

lcg_value (PHP 4 >= 4.0.0)

Combined linear congruential generator

float **lcg_value** (void) \linebreak

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 >= 4.0.0)

Natural logarithm

float **log** (float arg) \linebreak

Returns the natural logarithm of arg.

log10 (PHP 3, PHP 4 >= 4.0.0)

Base-10 logarithm

float **log10** (float arg) \linebreak

Returns the base-10 logarithm of *arg*.

log1p (PHP 4 >= 4.1.0)

Returns $\log(1 + \text{number})$, computed in a way that accurate even when the value of number is close to zero

float **log1p** (float number) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

max (PHP 3, PHP 4 >= 4.0.0)

Find highest value

mixed **max** (mixed arg1, mixed arg2, mixed argn) \linebreak

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 float, 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 float, all the values will be treated as floats, and a float is returned. If none of the values is a float, all of them will be treated as integers, and an integer is returned.

min (PHP 3, PHP 4 >= 4.0.0)

Find lowest value

number **min** (number arg1, number arg2 [, ...]) \linebreak number **min** (array numbers) \linebreak

min() returns the numerically lowest of the parameter values.

In the first variant, you need at least two parameters and **min()** returns the lowest of these values. You can compare an unlimited number of values.

In the second variant, **min()** returns the lowest value in *numbers*.

If one or more of the values is a float, all the values will be treated as floats, and a float is returned. If none of the values is a float, all of them will be treated as integers, and an integer is returned.

mt_getrandmax (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Show largest possible random value

int **mt_getrandmax** (void) \linebreak

Returns the maximum value that can be returned by a call to `mt_rand()`.

See also `mt_rand()`, `mt_srand()`, `rand()`, `srand()`, and `getrandmax()`.

mt_rand (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Generate a better random value

int **mt_rand** ([int min, int max]) \linebreak

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 >= 4.0.0)

Seed the better random number generator

void **mt_srand** (int seed) \linebreak

Seeds the random number generator with *seed*.

```
// seed with microseconds
function make_seed() {
    list($usec, $sec) = explode(' ', microtime());
    return (float) $sec + ((float) $usec * 100000);
}
mt_srand(make_seed());
$randval = mt_rand();
```

See also `mt_rand()`, `mt_getrandmax()`, `srand()`, `rand()`, and `getrandmax()`.

number_format (PHP 3, PHP 4 >= 4.0.0)

Format a number with grouped thousands

string **number_format** (float number [, int decimals [, string dec_point [, string thousands_sep]]) \linebreak

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.

Note: Only the first character of *thousands_sep* is used. For example, if you use `foo` as *thousands_sep* on the number 1000, **number_format()** will return `1f000`.

Example 1. number_format() Example

For instance, French notation usually use two decimals, comma (',') as decimal separator, and space (' ') as thousand separator. This is achieved with this line :

```
<?php

$number = 1234.56;

// english notation (default)
$english_format_number = number_format($number);
// 1,234.56

// French notation
$nombre_format_francais = number_format($number, 2, ',', ' ');
// 1 234,56

$number = 1234.5678;

// english notation without thousands separator
$english_format_number = number_format($number, 2, '.', '');
// 1234.56

?>
```

Note: See also: `sprintf()`, `printf()` and `sscanf()`.

octdec (PHP 3, PHP 4 >= 4.0.0)

Octal to decimal

int **octdec** (string *octal_string*) \linebreak

Returns the decimal equivalent of the octal number represented by the *octal_string* argument. The largest number that can be converted is 1777777777 or 2147483647 in decimal.

See also `decoct()`.

pi (PHP 3, PHP 4 >= 4.0.0)

Get value of pi

float **pi** (void) \linebreak

Returns an approximation of pi. The returned float has a precision based on the precision directive in `php.ini`, which defaults to 14. Also, you can use the `M_PI` constant which yields identical results to **pi()**.

```
echo pi(); // 3.1415926535898
echo M_PI; // 3.1415926535898
```

pow (PHP 3, PHP 4 >= 4.0.0)

Exponential expression

number **pow** (number *base*, number *exp*) \linebreak

Returns *base* raised to the power of *exp*. If possible, this function will return an integer.

If the power cannot be computed, a warning will be issued, and **pow()** will return `FALSE`.

Example 1. Some examples of pow()

```
<?php

var_dump( pow(2,8) ); // int(256)
echo pow(-1,20); // 1
echo pow(0, 0); // 1

echo pow(-1, 5.5); // error

?>
```

Warning

In PHP 4.0.6 and earlier **pow()** always returned a float, and did not issue warnings.

See also `exp()` and `sqrt()`.

rad2deg (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Converts the radian number to the equivalent number in degrees

```
float rad2deg ( float number ) \linebreak
```

This function converts *number* from radian to degrees.

See also `deg2rad()`.

rand (PHP 3, PHP 4 >= 4.0.0)

Generate a random value

```
int rand ( [int min, int max] ) \linebreak
```

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 `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 `rand (5 , 11)` to get a random number between 5 and 15.

See also `srand()`, `getrandmax()`, `mt_rand()`, `mt_srand()`, and `mt_getrandmax()`.

round (PHP 3, PHP 4 >= 4.0.0)

Rounds a float

float **round** (float *val* [, int *precision*]) \linebreak

Returns the rounded value of *val* to specified *precision* (number of digits after the decimal point). *precision* can also be negative or zero (default).

Caution

PHP doesn't handle strings like "12,300.2" correctly by default. See converting from strings.

```
$foo = round(3.4); // $foo == 3.0
$foo = round(3.5); // $foo == 4.0
$foo = round(3.6); // $foo == 4.0
$foo = round(3.6, 0); // equivalent with above

$foo = round(1.95583, 2); // $foo == 1.96

$foo = round(1241757, -3); // $foo == 1242000
```

Note: The *precision* parameter is only available in PHP 4.

See also `ceil()` and `floor()`.

sin (PHP 3, PHP 4 >= 4.0.0)

Sine

float **sin** (float *arg*) \linebreak

Returns the sine of *arg* in radians.

See also `cos()` and `tan()`.

sinh (PHP 4 >= 4.1.0)

Hyperbolic sine

float **sinh** (float *arg*) \linebreak

Returns the hyperbolic sine of *arg*, defined as $(\exp(\text{arg}) - \exp(-\text{arg}))/2$.

See also `sin()`, `asinh()`, `cos()` and `tan()`.

sqrt (PHP 3, PHP 4 >= 4.0.0)

Square root

float **sqrt** (float *arg*) \linebreak

Returns the square root of *arg*.

See also `pow()`.

srand (PHP 3, PHP 4 >= 4.0.0)

Seed the random number generator

void **srand** (int *seed*) \linebreak

Seeds the random number generator with *seed*.

```
// seed with microseconds
function make_seed() {
    list($usec, $sec) = explode(' ', microtime());
    return (float) $sec + ((float) $usec * 100000);
}
srand(make_seed());
$randval = rand();
```

See also `rand()`, `getrandmax()`, `mt_rand()`, `mt_srand()`, and `mt_getrandmax()`.

tan (PHP 3, PHP 4 >= 4.0.0)

Tangent

float **tan** (float *arg*) \linebreak

Returns the tangent of *arg* in radians.

See also `sin()` and `cos()`.

tanh (PHP 4 >= 4.1.0)

Hyperbolic tangent

float **tanh** (float *arg*) \linebreak

Returns the hyperbolic tangent of *arg*, defined as $\sinh(\text{arg}) / \cosh(\text{arg})$.

See also `tan()`, `atanh()`, `sin()` and `cos()`.

LIII. Multi-Byte String Functions

Introduction

There are many languages in which all characters can be expressed by single byte. Multi-byte character codes are used to express many characters for many languages. `mbstring` is developed to handle Japanese characters. However, many `mbstring` functions are able to handle character encoding other than Japanese.

A multi-byte character encoding represents single character with consecutive bytes. Some character encoding has shift(escape) sequences to start/end multi-byte character strings. Therefore, a multi-byte character string may be destroyed when it is divided and/or counted unless multi-byte character encoding safe method is used. This module provides multi-byte character safe string functions and other utility functions such as conversion functions.

Since PHP is basically designed for ISO-8859-1, some multi-byte character encoding does not work well with PHP. Therefore, it is important to set `mbstring.internal_encoding` to a character encoding that works with PHP.

PHP4 Character Encoding Requirements

- Per byte encoding
- Single byte characters in range of 00h-7fh which is compatible with ASCII
- Multi-byte characters without 00h-7fh

These are examples of internal character encoding that works with PHP and does NOT work with PHP.

Character encodings work with PHP:
ISO-8859-*, EUC-JP, UTF-8

Character encodings do NOT work with PHP:
JIS, SJIS

Character encoding, that does not work with PHP, may be converted with `mbstring`'s HTTP input/output conversion feature/function.

Note: SJIS should not be used for internal encoding unless the reader is familiar with parser/compiler, character encoding and character encoding issues.

Note: If you use database with PHP, it is recommended that you use the same character encoding

for both database and `internal` encoding for ease of use and better performance.

If you are using PostgreSQL, it supports character encoding that is different from backend character encoding. See the PostgreSQL manual for details.

How to Enable mbstring

`mbstring` is an extended module. You must enable module with `configure` script. Refer to the Install section for details.

The following configure options are related to `mbstring` module.

- `--enable-mbstring` : Enable `mbstring` functions. This option is required to use `mbstring` functions.
- `--enable-mbstr-enc-trans` : Enable HTTP input character encoding conversion using `mbstring` conversion engine. If this feature is enabled, HTTP input character encoding may be converted to `mbstring.internal_encoding` automatically.

HTTP Input and Output

HTTP input/output character encoding conversion may convert binary data also. Users are supposed to control character encoding conversion if binary data is used for HTTP input/output.

If `enctype` for HTML form is set to `multipart/form-data`, `mbstring` does not convert character encoding in POST data. If it is the case, strings are needed to be converted to internal character encoding.

- HTTP Input

There is no way to control HTTP input character conversion from PHP script. To disable HTTP input character conversion, it has to be done in `php.ini`.

Example 1. Disable HTTP input conversion in `php.ini`

```
;; Disable HTTP Input conversion
mbstring.http_input = pass
```

When using PHP as an Apache module, it is possible to override PHP ini setting per Virtual Host in `httpd.conf` or per directory with `.htaccess`. Refer to the Configuration section and Apache

Manual for details.

- HTTP Output

There are several ways to enable output character encoding conversion. One is using `php.ini`, another is using `ob_start()` with `mb_output_handler()` as `ob_start` callback function.

Note: For PHP3-i18n users, `mbstring`'s output conversion differs from PHP3-i18n. Character encoding is converted using output buffer.

Example 2. `php.ini` setting example

```
;; Enable output character encoding conversion for all PHP pages

;; Enable Output Buffering
output_buffering = On

;; Set mb_output_handler to enable output conversion
output_handler = mb_output_handler
```

Example 3. Script example

```
<?php

// Enable output character encoding conversion only for this page

// Set HTTP output character encoding to SJIS
mb_http_output('SJIS');

// Start buffering and specify "mb_output_handler" as
// callback function
ob_start('mb_output_handler');

?>
```

Supported Character Encoding

Currently, the following character encoding is supported by `mbstring` module. Character encoding may be specified for `mbstring` functions' `encoding` parameter.

The following character encoding is supported in this PHP extension :

UCS-4, UCS-4BE, UCS-4LE, UCS-2, UCS-2BE, UCS-2LE, UTF-32, UTF-32BE, UTF-32LE, UCS-2LE, UTF-16, UTF-16BE, UTF-16LE, UTF-8, UTF-7, ASCII, EUC-JP, SJIS, eucJP-win, SJIS-win, ISO-2022-JP, JIS, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-8859-10, ISO-8859-13, ISO-8859-14, ISO-8859-15, byte2be, byte2le, byte4be, byte4le, BASE64, 7bit, 8bit and UTF7-IMAP.

`php.ini` entry, which accepts encoding name, accepts "auto" and "pass" also. `mbstring` functions, which accepts encoding name, and accepts "auto".

If "pass" is set, no character encoding conversion is performed.

If "auto" is set, it is expanded to "ASCII, JIS, UTF-8, EUC-JP, SJIS".

See also `mb_detect_order()`

Note: "Supported character encoding" does not mean that it works as internal character code.

php.ini settings

- `mbstring.internal_encoding` defines default internal character encoding.
- `mbstring.http_input` defines default HTTP input character encoding.
- `mbstring.http_output` defines default HTTP output character encoding.
- `mbstring.detect_order` defines default character code detection order. See also `mb_detect_order()`.
- `mbstring.substitute_character` defines character to substitute for invalid character encoding.

Web Browsers are supposed to use the same character encoding when submitting form. However, browsers may not use the same character encoding. See `mb_http_input()` to detect character encoding

used by browsers.

If `enctype` is set to `multipart/form-data` in HTML forms, `mbstring` does not convert character encoding in POST data. The user must convert them in the script, if conversion is needed.

Although, browsers are smart enough to detect character encoding in HTML, `charset` is better to be set in HTTP header. Change `default_charset` according to character encoding.

Example 4. `php.ini` setting example

```

;; Set default internal encoding
;; Note: Make sure to use character encoding works with PHP
mbstring.internal_encoding = UTF-8 ; Set internal encoding to UTF-8

;; Set default HTTP input character encoding
;; Note: Script cannot change http_input setting.
mbstring.http_input       = pass    ; No conversion.
mbstring.http_input       = auto    ; Set HTTP input to auto
                             ; "auto" is expanded to "ASCII,JIS,UTF-8,EUC-JP,SJIS"
mbstring.http_input       = SJIS    ; Set HTTP2 input to SJIS
mbstring.http_input       = UTF-8,SJIS,EUC-JP ; Specify order

;; Set default HTTP output character encoding
mbstring.http_output      = pass    ; No conversion
mbstring.http_output      = UTF-8   ; Set HTTP output encoding to UTF-8

;; Set default character encoding detection order
mbstring.detect_order     = auto    ; Set detect order to auto
mbstring.detect_order     = ASCII,JIS,UTF-8,SJIS,EUC-JP ; Specify order

;; Set default substitute character
mbstring.substitute_character = 12307 ; Specify Unicode value
mbstring.substitute_character = none  ; Do not print character
mbstring.substitute_character = long  ; Long Example: U+3000,JIS+7E7E

```

Example 5. `php.ini` setting for EUC-JP users

```

;; Disable Output Buffering
output_buffering = Off

;; Set HTTP header charset
default_charset = EUC-JP

;; Set HTTP input encoding conversion to auto
mbstring.http_input = auto

```

```

;; Convert HTTP output to EUC-JP
mbstring.http_output = EUC-JP

;; Set internal encoding to EUC-JP
mbstring.internal_encoding = EUC-JP

;; Do not print invalid characters
mbstring.substitute_character = none

```

Example 6. php.ini setting for SJIS users

```

;; Enable Output Buffering
output_buffering = On

;; Set mb_output_handler to enable output conversion
output_handler = mb_output_handler

;; Set HTTP header charset
default_charset = Shift_JIS

;; Set http input encoding conversion to auto
mbstring.http_input = auto

;; Convert to SJIS
mbstring.http_output = SJIS

;; Set internal encoding to EUC-JP
mbstring.internal_encoding = EUC-JP

;; Do not print invalid characters
mbstring.substitute_character = none

```

Overload of PHP string functions by mbstring functions with multibyte support

Because almost PHP application written for language using single-byte character encoding, there are some difficulties for multibyte string handling including Japanese. Almost PHP string functions such as

substr() do not support multibyte string.

Multibyte extension (mbstring) has some PHP string functions with multibyte support (ex. substr() supports mb_substr()).

Multibyte extension (mbstring) also supports 'function overloading' to add multibyte string functionality without code modification. Using function overloading, some PHP string functions will be overloaded multibyte string functions. For example, mb_substr() is called instead of substr() if function overloading is enabled. Function overload makes easy to port application supporting only single-byte encoding for multibyte application.

mbstring.func_overload in php.ini should be set some positive value to use function overloading. The value should specify the category of overloading functions, should be set 1 to enable mail function overloading, 2 to enable string functions, 4 to regular expression functions. For example, if is set for 7, mail, strings, regex functions should be overloaded. The list of overloaded functions are shown in below.

Table 1. Functions to be overloaded

value of mbstring.func_overload	original function	overloaded function
1	mail()	mb_send_mail()
2	strlen()	mb_strlen()
2	strpos()	mb_strpos()
2	strrpos()	mb_strrpos()
2	substr()	mb_substr()
4	ereg()	mb_ereg()
4	eregi()	mb_eregi()
4	ereg_replace()	mb_ereg_replace()
4	eregi_replace()	mb_eregi_replace()
4	split()	mb_split()

Basics for Japanese multi-byte character

Most Japanese characters need more than 1 byte per character. In addition, several character encoding schemas are used under a Japanese environment. There are EUC-JP, Shift_JIS(SJIS) and ISO-2022-JP(JIS) character encoding. As Unicode becomes popular, UTF-8 is used also. To develop Web applications for a Japanese environment, it is important to use the character set for the task in hand, whether HTTP input/output, RDBMS and E-mail.

- Storage for a character can be up to six bytes
- A multi-byte character is usually twice of the width compared to single-byte characters. Wider characters are called "zen-kaku" - meaning full width, narrower characters are called "han-kaku" - meaning half width. "zen-kaku" characters are usually fixed width.
- Some character encoding defines shift(escape) sequence for entering/exiting multi-byte character

strings.

- ISO-2022-JP must be used for SMTP/NNTP.
- "i-mode" web site is supposed to use SJIS.

References

Multi-byte character encoding and its related issues are very complex. It is impossible to cover in sufficient detail here. Please refer to the following URLs and other resources for further readings.

- Unicode/UTF/UCS/etc

<http://www.unicode.org/>

- Japanese/Korean/Chinese character information

<ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/cjk.inf>

mb_convert_encoding (PHP 4 >= 4.0.6)

Convert character encoding

string **mb_convert_encoding** (string *str*, string *to-encoding* [, mixed *from-encoding*]) \linebreak

mb_convert_encoding() converts character encoding of string *str* from *from-encoding* to *to-encoding*.

str : String to be converted.

from-encoding is specified by character code name before conversion. it can be array or string - comma separated enumerated list. If it is not specified, the internal encoding will be used.

Example 1. mb_convert_encoding() example

```
/* Convert internal character encoding to SJIS */
$str = mb_convert_encoding($str, "SJIS");

/* Convert EUC-JP to UTF-7 */
$str = mb_convert_encoding($str, "UTF-7", "EUC-JP");

/* Auto detect encoding from JIS, eucjp-win, sjis-win, then convert str to UCS-2LE */
$str = mb_convert_encoding($str, "UCS-2LE", "JIS, eucjp-win, sjis-win");

/* "auto" is expanded to "ASCII,JIS,UTF-8,EUC-JP,SJIS" */
$str = mb_convert_encoding($str, "EUC-JP", "auto");
```

See also: mb_detect_order().

mb_convert_kana (PHP 4 >= 4.0.6)

Convert "kana" one from another ("zen-kaku" ,"han-kaku" and more)

string **mb_convert_kana** (string *str*, string *option* [, mixed *encoding*]) \linebreak

mb_convert_kana() performs "han-kaku" - "zen-kaku" conversion for string *str*. It returns converted string. This function is only useful for Japanese.

option is conversion option. Default value is "KV".

encoding is character encoding. If it is omitted, internal character encoding is used.

Applicable Conversion Options


```

option : Specify with conversion of following options. Default "KV"
"r" : Convert "zen-kaku" alphabets to "han-kaku"
"R" : Convert "han-kaku" alphabets to "zen-kaku"
"n" : Convert "zen-kaku" numbers to "han-kaku"
"N" : Convert "han-kaku" numbers to "zen-kaku"
"a" : Convert "zen-kaku" alphabets and numbers to "han-kaku"
"A" : Convert "zen-kaku" alphabets and numbers to "han-kaku"
(Characters included in "a", "A" options are
U+0021 - U+007E excluding U+0022, U+0027, U+005C, U+007E)
"s" : Convert "zen-kaku" space to "han-kaku" (U+3000 -> U+0020)
"S" : Convert "han-kaku" space to "zen-kaku" (U+0020 -> U+3000)
"k" : Convert "zen-kaku kata-kana" to "han-kaku kata-kana"
"K" : Convert "han-kaku kata-kana" to "zen-kaku kata-kana"
"h" : Convert "zen-kaku hira-gana" to "han-kaku kata-kana"
"H" : Convert "han-kaku kata-kana" to "zen-kaku hira-gana"
"c" : Convert "zen-kaku kata-kana" to "zen-kaku hira-gana"
"C" : Convert "zen-kaku hira-gana" to "zen-kaku kata-kana"
"V" : Collapse voiced sound notation and convert them into a character. Use with "K"

```

Example 1. mb_convert_kana() example

```

/* Convert all "kana" to "zen-kaku" "kata-kana" */
$str = mb_convert_kana($str, "KVC");

/* Convert "han-kaku" "kata-kana" to "zen-kaku" "kata-kana"
   and "zen-kaku" alpha-numeric to "han-kaku" */
$str = mb_convert_kana($str, "KVa");

```

mb_convert_variables (PHP 4 >= 4.0.6)

Convert character code in variable(s)

string **mb_convert_variables** (string to-encoding, mixed from-encoding, mixed vars) \linebreak

mb_convert_variables() convert character encoding of variables *vars* in encoding *from-encoding* to encoding *to-encoding*. It returns character encoding before conversion for success, FALSE for failure.

mb_convert_variables() join strings in Array or Object to detect encoding, since encoding detection tends to fail for short strings. Therefore, it is impossible to mix encoding in single array or object.

It *from-encoding* is specified by array or comma separated string, it tries to detect encoding from *from-coding*. When *encoding* is omitted, *detect_order* is used.

vars (3rd and larger) is reference to variable to be converted. String, Array and Object are accepted. **mb_convert_variables()** assumes all parameters have the same encoding.

Example 1. **mb_convert_variables()** example

```
/* Convert variables $post1, $post2 to internal encoding */
$interenc = mb_internal_encoding();
$inputenc = mb_convert_variables($interenc, "ASCII,UTF-8,SJIS-win", $post1, $post2);
```

mb_decode_mimeheader (PHP 4 >= 4.0.6)

Decode string in MIME header field

string **mb_decode_mimeheader** (string *str*) \linebreak

mb_decode_mimeheader() decodes encoded-word string *str* in MIME header.

It returns decoded string in internal character encoding.

See also **mb_encode_mimeheader()**.

mb_decode_numericentity (PHP 4 >= 4.0.6)

Decode HTML numeric string reference to character

string **mb_decode_numericentity** (string *str*, array *convmap* [, string *encoding*]) \linebreak

Convert numeric string reference of string *str* in specified block to character. It returns converted string.

array is array to specifies code area to convert.

encoding is character encoding. If it is omitted, internal character encoding is used.

Example 1. *convmap* example

```
$convmap = array (
    int start_code1, int end_code1, int offset1, int mask1,
    int start_code2, int end_code2, int offset2, int mask2,
    .....
    int start_codeN, int end_codeN, int offsetN, int maskN );
```

```
// Specify Unicode value for start_codeN and end_codeN
// Add offsetN to value and take bit-wise 'AND' with maskN,
// then convert value to numeric string reference.
```

See also: `mb_encode_numericentity()`.

mb_detect_encoding (PHP 4 >= 4.0.6)

Detect character encoding

string **mb_detect_encoding** (string *str* [, mixed *encoding-list*] \linebreak

mb_detect_encoding() detects character encoding in string *str*. It returns detected character encoding.

encoding-list is list of character encoding. Encoding order may be specified by array or comma separated list string.

If *encoding_list* is omitted, *detect_order* is used.

Example 1. mb_detect_encoding() example

```
/* Detect character encoding with current detect_order */
echo mb_detect_encoding($str);

/* "auto" is expanded to "ASCII,JIS,UTF-8,EUC-JP,SJIS" */
echo mb_detect_encoding($str, "auto");

/* Specify encoding_list character encoding by comma separated list */
echo mb_detect_encoding($str, "JIS, eucjp-win, sjis-win");

/* Use array to specify encoding_list */
$array[] = "ASCII";
$array[] = "JIS";
$array[] = "EUC-JP";
echo mb_detect_encoding($str, $array);
```

See also: `mb_detect_order()`.

mb_detect_order (PHP 4 >= 4.0.6)

Set/Get character encoding detection order

array **mb_detect_order** ([mixed encoding-list]) \linebreak

mb_detect_order() sets automatic character encoding detection order to *encoding-list*. It returns TRUE for success, FALSE for failure.

encoding-list is array or comma separated list of character encoding. ("auto" is expanded to "ASCII, JIS, UTF-8, EUC-JP, SJIS")

If *encoding-list* is omitted, it returns current character encoding detection order as array.

This setting affects mb_detect_encoding() and mb_send_mail().

Note: *mbstring* currently implements following encoding detection filters. If there is a invalid byte sequence for following encoding, encoding detection will fail.

Note: UTF-8, UTF-7, ASCII, EUC-JP,SJIS, eucJP-win, SJIS-win, JIS, ISO-2022-JP

For ISO-8859-*, *mbstring* always detects as ISO-8859-*.

For UTF-16, UTF-32, UCS2 and UCS4, encoding detection will fail always.

Example 1. Useless detect order example

```

; Always detect as ISO-8859-1
detect_order = ISO-8859-1, UTF-8

; Always detect as UTF-8, since ASCII/UTF-7 values are
; valid for UTF-8
detect_order = UTF-8, ASCII, UTF-7

```

Example 2. mb_detect_order() examples

```

/* Set detection order by enumerated list */
mb_detect_order("eucjp-win,sjis-win,UTF-8");

/* Set detection order by array */
$array[] = "ASCII";
$array[] = "JIS";
$array[] = "EUC-JP";
mb_detect_order($array);

/* Display current detection order */
echo implode(", ", mb_detect_order());

```

See also `mb_internal_encoding()`, `mb_http_input()`, `mb_http_output()` `mb_send_mail()`.

mb_encode_mimeheader (PHP 4 >= 4.0.6)

Encode string for MIME header

string **mb_encode_mimeheader** (string *str* [, string *charset* [, string *transfer-encoding* [, string *linefeed*]]) \linebreak

mb_encode_mimeheader() converts string *str* to encoded-word for header field. It returns converted string in ASCII encoding.

charset is character encoding name. Default is `ISO-2022-JP`.

transfer-encoding is transfer encoding. It should be one of "B" (Base64) or "Q" (Quoted-Printable). Default is "B".

linefeed is end of line marker. Default is `"\r\n"` (CRLF).

Example 1. mb_convert_kana() example

```
$name = ""; // kanji
$mbox = "kru";
$doma = "gtinn.mon";
$addr = mb_encode_mimeheader($name, "UTF-7", "Q") . " <" . $mbox . "@" . $doma . ">";
echo $addr;
```

See also `mb_decode_mimeheader()`.

mb_encode_numericentity (PHP 4 >= 4.0.6)

Encode character to HTML numeric string reference

string **mb_encode_numericentity** (string *str*, array *convmap* [, string *encoding*]) \linebreak

mb_encode_numericentity() converts specified character codes in string *str* from HTML numeric character reference to character code. It returns converted string.

array is array specifies code area to convert.

encoding is character encoding.

Example 1. *convmap* example

```
$convmap = array (
    int start_code1, int end_code1, int offset1, int mask1,
    int start_code2, int end_code2, int offset2, int mask2,
    .....
    int start_codeN, int end_codeN, int offsetN, int maskN );
// Specify Unicode value for start_codeN and end_codeN
// Add offsetN to value and take bit-wise 'AND' with maskN, then
// it converts value to numeric string reference.
```

Example 2. *mb_encode_numericentity()* example

```
/* Convert Left side of ISO-8859-1 to HTML numeric character reference */
$convmap = array(0x80, 0xff, 0, 0xff);
$str = mb_encode_numericentity($str, $convmap, "ISO-8859-1");

/* Convert user defined SJIS-win code in block 95-104 to numeric
string reference */
$convmap = array(
    0xe000, 0xe03e, 0x1040, 0xffff,
    0xe03f, 0xe0bb, 0x1041, 0xffff,
    0xe0bc, 0xe0fa, 0x1084, 0xffff,
    0xe0fb, 0xe177, 0x1085, 0xffff,
    0xe178, 0xe1b6, 0x10c8, 0xffff,
    0xe1b7, 0xe233, 0x10c9, 0xffff,
    0xe234, 0xe272, 0x110c, 0xffff,
    0xe273, 0xe2ef, 0x110d, 0xffff,
    0xe2f0, 0xe32e, 0x1150, 0xffff,
    0xe32f, 0xe3ab, 0x1151, 0xffff );
$str = mb_encode_numericentity($str, $convmap, "sjis-win");
```

See also: *mb_decode_numericentity()*.

mb_ereg (PHP 4 >= 4.2.0)

Regular expression match with multibyte support

int **mb_ereg** (string pattern, string string [, array regs]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg() executes the regular expression match with multibyte support, and returns 1 if matches are found. If the optional third parameter was specified, the function returns the byte length of matched part, and the array *regs* will contain the substring of matched string. The function returns 1 if it matches with the empty string. If no match is found or an error happens, **FALSE** will be returned.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_eregi()`

mb_ereg_match (PHP 4 >= 4.2.0)

Regular expression match for multibyte string

bool **mb_ereg_match** (string pattern, string string [, string option]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_match() returns **TRUE** if *string* matches regular expression *pattern*, **FALSE** if not.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg()`.

mb_ereg_replace (PHP 4 >= 4.2.0)

Replace regular expression with multibyte support

string **mb_ereg_replace** (string pattern, string replacement, string string [, array option]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_replace() scans *string* for matches to *pattern*, then replaces the matched text with *replacement* and returns the result string or `FALSE` on error. Multibyte character can be used in *pattern*.

Matching condition can be set by *option* parameter. If `i` is specified for this parameter, the case will be ignored. If `x` is specified, white space will be ignored. If `m` is specified, match will be executed in multiline mode and line break will be included in `.`. If `p` is specified, match will be executed in POSIX mode, line break will be considered as normal character. If `e` is specified, *replacement* string will be evaluated as PHP expression.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_eregi_replace()`.

mb_ereg_search (PHP 4 >= 4.2.0)

Multibyte regular expression match for predefined multibyte string

bool **mb_ereg_search** ([string pattern [, string option]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search() returns `TRUE` if the multibyte string matches with the regular expression, `FALSE` for otherwise. The string for matching is set by `mb_ereg_search_init()`. If *pattern* is not specified, the previous one is used.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_init()`.

mb_ereg_search_getpos (PHP 4 >= 4.2.0)

Returns start point for next regular expression match

array **mb_ereg_search_getpos** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_getpos() returns the point to start regular expression match for `mb_ereg_search()`, `mb_ereg_search_pos()`, `mb_ereg_search_regs()`. The position is represented by bytes from the head of string.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_setpos()`.

mb_ereg_search_getregs (PHP 4 >= 4.2.0)

Retrieve the result from the last multibyte regular expression match

array **mb_ereg_search_getregs** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_getregs() returns an array including the sub-string of matched part by last `mb_ereg_search()`, `mb_ereg_search_pos()`, `mb_ereg_search_regs()`. If there are some matches, the first element will have the matched sub-string, the second element will have the first part grouped with brackets, the third element will have the second part grouped with brackets, and so on. It returns `FALSE` on error;

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_init()`.

mb_ereg_search_init (PHP 4 >= 4.2.0)

Setup string and regular expression for multibyte regular expression match

array **mb_ereg_search_init** (string *string* [, string *pattern* [, string *option*]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_init() sets *string* and *pattern* for multibyte regular expression. These values are used for `mb_ereg_search()`, `mb_ereg_search_pos()`, `mb_ereg_search_regs()`. It returns `TRUE` for success, `FALSE` for error.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_regs()`.

mb_ereg_search_pos (PHP 4 >= 4.2.0)

Return position and length of matched part of multibyte regular expression for predefined multibyte string

array **mb_ereg_search_pos** ([string *pattern* [, string *option*]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_pos() returns an array including position of matched part for multibyte regular expression. The first element of the array will be the beginning of matched part, the second element will be length (bytes) of matched part. It returns `FALSE` on error.

The string for match is specified by `mb_ereg_search_init()`. If it is not specified, the previous one will be used.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_init()`.

mb_ereg_search_regs (PHP 4 >= 4.2.0)

Returns the matched part of multibyte regular expression

array **mb_ereg_search_regs** ([string pattern [, string option]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_regs() executes the multibyte regular expression match, and if there are some matched part, it returns an array including substring of matched part as first element, the first grouped part with brackets as second element, the second grouped part as third element, and so on. It returns `FALSE` on error.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_init()`.

mb_ereg_search_setpos (PHP 4 >= 4.2.0)

Set start point of next regular expression match

array **mb_ereg_search_setpos** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_ereg_search_setpos() sets the starting point of match for `mb_ereg_search()`.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_search_init()`.

mb_eregi (PHP 4 >= 4.2.0)

Regular expression match ignoring case with multibyte support

int **mb_eregi** (string pattern, string string [, array regs]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_eregi() executes the regular expression match with multibyte support, and returns 1 if matches are found. This function ignore case. If the optional third parameter was specified, the function returns the byte length of matched part, and the array *regs* will contain the substring of matched string. The function returns 1 if it matches with the empty string. If no match found or error happen, `FALSE` will be returned.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg()`.

mb_eregi_replace (PHP 4 >= 4.2.0)

Replace regular expression with multibyte support ignoring case

string **mb_eregi_replace** (string pattern, string replace, string string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`mb_eregi_replace()` scans *string* for matches to *pattern*, then replaces the matched text with *replacement* and returns the result string or `FALSE` on error. Multibyte character can be used in *pattern*. The case will be ignored.

The internal encoding or the character encoding specified in `mb_regex_encoding()` will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_regex_encoding()`, `mb_ereg_replace()`.

mb_get_info (PHP 4 >= 4.2.0)

Get internal settings of mbstring

string **mb_get_info** ([string type]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`mb_get_info()` returns internal setting parameter of mbstring.

If *type* isn't specified or is specified to "all", an array having the elements "internal_encoding", "http_output", "http_input", "func_overload" will be returned.

If *type* is specified for "http_output", "http_input", "internal_encoding", "func_overload", the specified setting parameter will be returned.

See also `mb_internal_encoding()`, `mb_http_output()`.

mb_http_input (PHP 4 >= 4.0.6)

Detect HTTP input character encoding

string **mb_http_input** ([string type]) \linebreak

mb_http_input() returns result of HTTP input character encoding detection.

type: Input string specifies input type. "G" for GET, "P" for POST, "C" for COOKIE. If type is omitted, it returns last input type processed.

Return Value: Character encoding name. If **mb_http_input()** does not process specified HTTP input, it returns `FALSE`.

See also `mb_internal_encoding()`, `mb_http_output()`, `mb_detect_order()`.

mb_http_output (PHP 4 >= 4.0.6)

Set/Get HTTP output character encoding

string **mb_http_output** ([string encoding]) \linebreak

If *encoding* is set, **mb_http_output()** sets HTTP output character encoding to *encoding*. Output after this function is converted to *encoding*. **mb_http_output()** returns `TRUE` for success and `FALSE` for failure.

If *encoding* is omitted, **mb_http_output()** returns current HTTP output character encoding.

See also `mb_internal_encoding()`, `mb_http_input()`, `mb_detect_order()`.

mb_internal_encoding (PHP 4 >= 4.0.6)

Set/Get internal character encoding

string **mb_internal_encoding** ([string encoding]) \linebreak

mb_internal_encoding() sets internal character encoding to *encoding*. If parameter is omitted, it returns current internal encoding.

encoding is used for HTTP input character encoding conversion, HTTP output character encoding conversion and default character encoding for string functions defined by mbstring module.

encoding: Character encoding name

Return Value: If *encoding* is set, **mb_internal_encoding()** returns `TRUE` for success, otherwise returns `FALSE`. If *encoding* is omitted, it returns current character encoding name.

Example 1. mb_internal_encoding() example

```

/* Set internal character encoding to UTF-8 */
mb_internal_encoding("UTF-8");

/* Display current internal character encoding */
echo mb_internal_encoding();

```

See also `mb_http_input()`, `mb_http_output()`, `mb_detect_order()`.

mb_language (PHP 4 >= 4.0.6)

Set/Get current language

string **mb_language** ([string language]) \linebreak

mb_language() sets language. If *language* is omitted, it returns current language as string.

language setting is used for encoding e-mail messages. Valid languages are "Japanese", "ja", "English", "en" and "uni" (UTF-8). `mb_send_mail()` uses this setting to encode e-mail.

Language and its setting is ISO-2022-JP/Base64 for Japanese, UTF-8/Base64 for uni, ISO-8859-1/quoted printable for English.

Return Value: If *language* is set and *language* is valid, it returns TRUE. Otherwise, it returns FALSE. When *language* is omitted, it returns language name as string. If no language is set previously, it returns FALSE.

See also `mb_send_mail()`.

mb_output_handler (PHP 4 >= 4.0.6)

Callback function converts character encoding in output buffer

string **mb_output_handler** (string contents, int status) \linebreak

mb_output_handler() is `ob_start()` callback function. **mb_output_handler()** converts characters in output buffer from internal character encoding to HTTP output character encoding.

4.1.0 or later version, this handler adds charset HTTP header when following conditions are met:

- Does not set `Content-Type` by `header()`
- Default MIME type begins with `text/`

- `http_output` setting is other than `pass`

contents : Output buffer contents

status : Output buffer status

Return Value: String converted

Example 1. `mb_output_handler()` example

```
mb_http_output( "UTF-8" );
ob_start( "mb_output_handler" );
```

Note: If you want to output some binary data such as image from PHP script, you must set output encoding to "pass" using `mb_http_output()`.

See also `ob_start()`.

`mb_parse_str` (PHP 4 >= 4.0.6)

Parse GET/POST/COOKIE data and set global variable

boolean **`mb_parse_str`** (string *encoded_string* [, array *result*]) \linebreak

`mb_parse_str()` parses GET/POST/COOKIE data and sets global variables. Since PHP does not provide raw POST/COOKIE data, it can only be used for GET data for now. It parses URL encoded data, detects encoding, converts coding to internal encoding and sets values to *result* array or global variables.

encoded_string: URL encoded data.

result: Array contains decoded and character encoding converted values.

Return Value: It returns `TRUE` for success or `FALSE` for failure.

See also `mb_detect_order()`, `mb_internal_encoding()`.

`mb_preferred_mime_name` (PHP 4 >= 4.0.6)

Get MIME charset string

string **`mb_preferred_mime_name`** (string *encoding*) \linebreak

mb_preferred_mime_name() returns MIME charset string for character encoding *encoding*. It returns charset string.

Example 1. mb_preferred_mime_string() example

```
$outputenc = "sjis-win";
mb_http_output($outputenc);
ob_start("mb_output_handler");
header("Content-Type: text/html; charset=" . mb_preferred_mime_name($outputenc));
```

mb_regex_encoding (PHP 4 >= 4.2.0)

Returns current encoding for multibyte regex as string

string **mb_regex_encoding** ([string encoding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_regex_encoding() returns the character encoding used by multibyte regex functions.

If the optional parameter *encoding* is specified, it is set to the character encoding for multibyte regex. The default value is the internal character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: `mb_internal_encoding()`, `mb_ereg()`

mb_send_mail (PHP 4 >= 4.0.6)

Send encoded mail.

boolean **mb_send_mail** (string to, string subject, string message [, string additional_headers [, string additional_parameter]]) \linebreak

mb_send_mail() sends email. Headers and message are converted and encoded according to `mb_language()` setting. **mb_send_mail()** is wrapper function of `mail()`. See `mail()` for details.

to is mail addresses send to. Multiple recipients can be specified by putting a comma between each address in *to*. This parameter is not automatically encoded.

subject is subject of mail.

message is mail message.

additional_headers is inserted at the end of the header. This is typically used to add extra headers. Multiple extra headers are separated with a newline ("\n").

additional_parameter is a MTA command line parameter. It is useful when setting the correct Return-Path header when using sendmail.

Returns TRUE on success, FALSE on failure.

See also mail(), mb_encode_mimeheader(), and mb_language().

mb_split (PHP 4 >= 4.2.0)

Split multibyte string using regular expression

array **mb_split** (string *pattern*, string *string* [, int *limit*]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

mb_split() split multibyte *string* using regular expression *pattern* and returns the result as an array.

If optional parameter *limit* is specified, it will be split in *limit* elements as maximum.

The internal encoding or the character encoding specified in mb_regex_encoding() will be used as character encoding.

Note: This function is supported in PHP 4.2.0 or higher.

See also: mb_regex_encoding(), mb_ereg().

mb_strcut (PHP 4 >= 4.0.6)

Get part of string

string **mb_strcut** (string *str*, int *start* [, int *length* [, string *encoding*]]) \linebreak

mb_strcut() returns the portion of *str* specified by the *start* and *length* parameters.

mb_strcut() performs equivalent operation as `mb_substr()` with different method. If *start* position is multi-byte character's second byte or larger, it starts from first byte of multi-byte character.

It subtracts string from *str* that is shorter than *length* AND character that is not part of multi-byte string or not being middle of shift sequence.

encoding is character encoding. If it is not set, internal character encoding is used.

See also `mb_substr()`, `mb_internal_encoding()`.

mb_strimwidth (PHP 4 >= 4.0.6)

Get truncated string with specified width

string **mb_strimwidth** (string *str*, int *start*, int *width*, string *trimmarker* [, string *encoding*]) \linebreak

mb_strimwidth() truncates string *str* to specified *width*. It returns truncated string.

If *trimmarker* is set, *trimmarker* is appended to return value.

start is start position offset. Number of characters from the beginning of string. (First character is 0)

trimmarker is string that is added to the end of string when string is truncated.

encoding is character encoding. If it is omitted, internal encoding is used.

Example 1. mb_strimwidth() example

```
$str = mb_strimwidth($str, 0, 40, "..>");
```

See also: `mb_strwidth()`, `mb_internal_encoding()`.

mb_strlen (PHP 4 >= 4.0.6)

Get string length

string **mb_strlen** (string *str* [, string *encoding*]) \linebreak

mb_strlen() returns number of characters in string *str* having character encoding *encoding*. A multi-byte character is counted as 1.

encoding is character encoding for *str*. If *encoding* is omitted, internal character encoding is used.

See also `mb_internal_encoding()`, `strlen()`.

mb_strpos (PHP 4 >= 4.0.6)

Find position of first occurrence of string in a string

```
int mb_strpos ( string haystack, string needle [, int offset [, string encoding]]) \linebreak
```

mb_strpos() returns the numeric position of the first occurrence of *needle* in the *haystack* string. If *needle* is not found, it returns `FALSE`.

mb_strpos() performs multi-byte safe `strpos()` operation based on number of characters. *needle* position is counted from the beginning of the *haystack*. First character's position is 0. Second character position is 1, and so on.

If *encoding* is omitted, internal character encoding is used. `mb_strpos()` accepts *string* for *needle* where `strpos()` accepts only character.

offset is search offset. If it is not specified, 0 is used.

encoding is character encoding name. If it is omitted, internal character encoding is used.

See also `mb_strpos()`, `mb_internal_encoding()`, `strpos()`

mb_strrpos (PHP 4 >= 4.0.6)

Find position of last occurrence of a string in a string

```
int mb_strrpos ( string haystack, string needle [, string encoding]) \linebreak
```

mb_strrpos() returns the numeric position of the last occurrence of *needle* in the *haystack* string. If *needle* is not found, it returns `FALSE`.

mb_strrpos() performs multi-byte safe `strrpos()` operation based on number of characters. *needle* position is counted from the beginning of *haystack*. First character's position is 0. Second character position is 1.

If *encoding* is omitted, internal encoding is assumed. **mb_strrpos()** accepts *string* for *needle* where `strrpos()` accepts only character.

encoding is character encoding. If it is not specified, internal character encoding is used.

See also `mb_strpos()`, `mb_internal_encoding()`, `strrpos()`.

mb_strwidth (PHP 4 >= 4.0.6)

Return width of string

```
int mb_strwidth ( string str [, string encoding]) \linebreak
```

mb_strwidth() returns width of string *str*.

Multi-byte character usually twice of width compare to single byte character.

Character width		
U+0000 - U+0019		0
U+0020 - U+1FFF		1
U+2000 - U+FF60		2
U+FF61 - U+FF9F		1
U+FFA0 -		2

encoding is character encoding. If it is omitted, internal encoding is used.

See also: `mb_striwidth()`, `mb_internal_encoding()`.

mb_substitute_character (PHP 4 >= 4.0.6)

Set/Get substitution character

mixed **mb_substitute_character** ([mixed *substchar*] \linebreak

mb_substitute_character() specifies substitution character when input character encoding is invalid or character code is not exist in output character encoding. Invalid characters may be substituted `NULL`(no output), string or integer value (Unicode character code value).

This setting affects `mb_detect_encoding()` and `mb_send_mail()`.

substchar : Specify Unicode value as integer or specify as string as follows

- "none" : no output
- "long" : Output character code value (Example: U+3000,JIS+7E7E)

Return Value: If *substchar* is set, it returns `TRUE` for success, otherwise returns `FALSE`. If *substchar* is not set, it returns Unicode value or "none"/"long".

Example 1. mb_substitute_character() example

```
/* Set with Unicode U+3013 (GETA MARK) */
mb_substitute_character(0x3013);

/* Set hex format */
mb_substitute_character("long");

/* Display current setting */
echo mb_substitute_character();
```

mb_substr (PHP 4 >= 4.0.6)

Get part of string

string **mb_substr** (string *str*, int *start* [, int *length* [, string *encoding*]]) \linebreak

mb_substr() returns the portion of *str* specified by the *start* and *length* parameters.

mb_substr() performs multi-byte safe substr() operation based on number of characters. Position is counted from the beginning of *str*. First character's position is 0. Second character position is 1, and so on.

If *encoding* is omitted, internal encoding is assumed.

encoding is character encoding. If it is omitted, internal character encoding is used.

See also mb_strcut(), mb_internal_encoding().

LIV. 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 pluggable 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 recurring events.

With libmcal, central calendar servers can be accessed, 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. For weekdays :

- MCAL_SUNDAY
- MCAL_MONDAY
- MCAL_TUESDAY
- MCAL_WEDNESDAY
- MCAL_THURSDAY
- MCAL_FRIDAY
- MCAL_SATURDAY

For recurrence :

- MCAL_RECUR_NONE
- MCAL_RECUR_DAILY
- MCAL_RECUR_WEEKLY
- MCAL_RECUR_MONTHLY_MDAY
- MCAL_RECUR_MONTHLY_WDAY
- MCAL_RECUR_YEARLY

For months :

- MCAL_JANUARY
- MCAL_FEBRUARY
- MCAL_MARCH
- MCAL_APRIL
- MCAL_MAY
- MCAL_JUNE
- MCAL_JULY
- MCAL_AUGUST
- MCAL_SEPTEMBER
- MCAL_OCTOBER
- MCAL_NOVEMBER
- 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_append_event (PHP 4 >= 4.0.0)

Store a new event into an MCAL calendar

int **mcal_append_event** (int mcal_stream) \linebreak

mcal_append_event() Stores the global event into an MCAL calendar for the given stream.

Returns the id of the newly inserted event.

mcal_close (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Close an MCAL stream

int **mcal_close** (int mcal_stream, int flags) \linebreak

Closes the given mcal stream.

mcal_create_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Create a new MCAL calendar

string **mcal_create_calendar** (int stream, string calendar) \linebreak

Creates a new calendar named *calendar*.

mcal_date_compare (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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) \linebreak

mcal_date_compare() Compares the two given dates, returns <0, 0, >0 if a<b, a==b, a>b respectively.

mcal_date_valid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns TRUE if the given year, month, day is a valid date

int **mcal_date_valid** (int year, int month, int day) \linebreak

mcal_date_valid() Returns TRUE if the given year, month and day is a valid date, FALSE if not.

mcal_day_of_week (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the day of the week of the given date

int **mcal_day_of_week** (int year, int month, int day) \linebreak

mcal_day_of_week() returns the day of the week of the given date. Possible return values range from 0 for Sunday through 6 for Saturday.

mcal_day_of_year (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the day of the year of the given date

int **mcal_** (int year, int month, int day) \linebreak

mcal_day_of_year() returns the day of the year of the given date.

mcal_days_in_month (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the number of days in the given month

int **mcal_days_in_month** (int month, int leap year) \linebreak

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_delete_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Delete an MCAL calendar

string **mcal_delete_calendar** (int stream, string calendar) \linebreak

Deletes the calendar named *calendar*.

mcal_delete_event (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Delete an event from an MCAL calendar

int **mcal_delete_event** (int mcal_stream [, int event_id]) \linebreak

mcal_delete_event() deletes the calendar event specified by the *event_id*.

Returns TRUE.

mcald_event_add_attribute (PHP 3>= 3.0.15, PHP 4 >= 4.0.0)

Adds an attribute and a value to the streams global event structure

```
void mcald_event_add_attribute ( int stream, string attribute, string value) \linebreak
```

mcald_event_add_attribute() adds an attribute to the stream's global event structure with the value given by "value".

mcald_event_init (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Initializes a streams global event structure

```
int mcald_event_init ( int stream) \linebreak
```

mcald_event_init() initializes a streams global event structure. this effectively sets all elements of the structure to 0, or the default settings.

Returns TRUE.

mcald_event_set_alarm (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the alarm of the streams global event structure

```
int mcald_event_set_alarm ( int stream, int alarm) \linebreak
```

mcald_event_set_alarm() sets the streams global event structure's alarm to the given minutes before the event.

Returns TRUE.

mcald_event_set_category (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the category of the streams global event structure

```
int mcald_event_set_category ( int stream, string category) \linebreak
```

mcald_event_set_category() sets the streams global event structure's category to the given string.

Returns TRUE.

mcal_event_set_class (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the class of the streams global event structure

```
int mcal_event_set_class ( int stream, int class) \linebreak
```

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_event_set_description (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the description of the streams global event structure

```
int mcal_event_set_description ( int stream, string description) \linebreak
```

mcal_event_set_description() sets the streams global event structure's description to the given string.

Returns TRUE.

mcal_event_set_end (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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]]]]) \linebreak
```

mcal_event_set_end() sets the streams global event structure's end date and time to the given values.

Returns TRUE.

mcal_event_set_recur_daily (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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) \linebreak
```

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_monthly_mday (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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) \linebreak

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 >= 4.0.0)

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) \linebreak

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_none (PHP 3>= 3.0.15, PHP 4 >= 4.0.0)

Sets the recurrence of the streams global event structure

int mcal_event_set_recur_none (int stream) \linebreak

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_weekly (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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) \linebreak

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_yearly (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

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) \linebreak

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.

mcald_event_set_start (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the start date and time of the streams global event structure

```
int mcald_event_set_start ( int stream, int year, int month [, int day [, int hour [, int min [, int sec]]]]) \linebreak
```

mcald_event_set_start() sets the streams global event structure's start date and time to the given values.

Returns TRUE.

mcald_event_set_title (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Sets the title of the streams global event structure

```
int mcald_event_set_title ( int stream, string title) \linebreak
```

mcald_event_set_title() sets the streams global event structure's title to the given string.

Returns TRUE.

mcald_expunge (unknown)

Deletes all events marked for being expunged.

```
int mcald_expunge ( int stream) \linebreak
```

mcald_expunge() Deletes all events which have been previously marked for deletion.

mcald_fetch_current_stream_event (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns an object containing the current streams event structure

```
object mcald_fetch_current_stream_event ( int stream) \linebreak
```

mcald_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

mcald_fetch_event (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Fetches an event from the calendar stream

object **mcald_fetch_event** (int mcald_stream, int event_id [, int options]) \linebreak

mcald_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

The possible values for recur_type are:

- 0 - Indicates that this event does not recur
- 1 - This event recurs daily
- 2 - This event recurs on a weekly basis
- 3 - This event recurs monthly on a specific day of the month (e.g. the 10th of the month)
- 4 - This event recurs monthly on a sequenced day of the week (e.g. the 3rd Saturday)
- 5 - This event recurs on an annual basis

mcald_is_leap_year (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns if the given year is a leap year or not

int **mcald_is_leap_year** (int year) \linebreak

mcald_is_leap_year() returns 1 if the given year is a leap year, 0 if not.

mcald_list_alarms (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return a list of events that has an alarm triggered at the given datetime

array **mcald_list_alarms** (int mcald_stream [, int begin_year [, int begin_month [, int begin_day [, int end_year [, int end_month [, int end_day]]]]]) \linebreak

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.

mcald_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.

mcald_list_events (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return a list of IDs for a date or a range of dates.

array **mcald_list_events** (int mcald_stream, object begin_date [, object end_date]) \linebreak

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.

mcald_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.

mcald_next_recurrence (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns the next recurrence of the event

int **mcald_next_recurrence** (int stream, int weekstart, array next) \linebreak

mcald_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.

mcald_open (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Opens up an MCAL connection

int **mcald_open** (string calendar, string username, string password [, int options]) \linebreak

Returns an MCAL stream on success, FALSE on error.

mcald_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.

mcald_popen (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Opens up a persistent MCAL connection

int **mcald_popen** (string calendar, string username, string password [, int options]) \linebreak

Returns an MCAL stream on success, FALSE on error.

mcald_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.

mcalf_rename_calendar (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Rename an MCAL calendar

```
string mcalf_rename_calendar ( int stream, string old_name, string new_name) \linebreak
```

Renames the calendar *old_name* to *new_name*.

mcalf_reopen (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Reopens an MCAL connection

```
int mcalf_reopen ( string calendar [, int options]) \linebreak
```

Reopens an MCAL stream to a new calendar.

mcalf_reopen() reopens an MCAL connection to the specified *calendar* store. If the optional *options* is specified, passes the *options* to that mailbox also.

mcalf_snooze (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Turn off an alarm for an event

```
int mcalf_snooze ( int id) \linebreak
```

mcalf_snooze() turns off an alarm for a calendar event specified by the id.

Returns TRUE.

mcalf_store_event (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Modify an existing event in an MCAL calendar

```
int mcalf_store_event ( int mcalf_stream) \linebreak
```

mcalf_store_event() Stores the modifications to the current global event for the given stream.

Returns the event id of the modified event on success and FALSE on error.

mcalf_time_valid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Returns TRUE if the given year, month, day is a valid time

int **mcald_time_valid** (int hour, int minutes, int seconds) \linebreak

mcald_time_valid() Returns `TRUE` if the given hour, minutes and seconds is a valid time, `FALSE` if not.

mcald_week_of_year (PHP 4 >= 4.0.0)

Returns the week number of the given date

int **mcald_week_of_year** (int day, int month, int year) \linebreak

LV. Mcrypt Encryption Functions

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, TWOFISH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes. Additionally, it supports RC6 and IDEA which are considered "non-free".

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`.

Requirements

These functions work using `mcrypt` (<http://mcrypt.hellug.gr/>).

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`.

Installation

To use it, download `libmcrypt-x.x.tar.gz` from here (<http://mcrypt.hellug.gr/>) 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`.

Runtime Configuration

Resource types

This extension does not define any resource types.

Predefined constants

`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. Below you find a list with all supported encryption modes together with the constants that are defines for the encryption mode. For a more complete reference and discussion see *Applied Cryptography* by Schneier (ISBN 0-471-11709-9).

- `MCRYPT_MODE_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.
- `MCRYPT_MODE_CBC` (cipher block chaining) is especially suitable for encrypting files where the security is increased over ECB significantly.
- `MCRYPT_MODE_CFB` (cipher feedback) is the best mode for encrypting byte streams where single

bytes must be encrypted.

- `MCRYPT_MODE_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.
- `MCRYPT_MODE_NOFB` (output feedback, in nbit) is comparable to OFB, but more secure because it operates on the block size of the algorithm.
- `MCRYPT_MODE_STREAM` is an extra mode to include some stream algorithms like WAKE or RC4.

Here is a list of ciphers which are currently supported by the `mcrypt` extension. 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().

- 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)

You must (in CFB and OFB mode) or can (in CBC mode) supply an initialization vector (IV) to the respective cipher 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_cbc (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Encrypt/decrypt data in CBC mode

```
string mcrypt_cbc ( int cipher, string key, string data, int mode [, string iv]) \linebreak string mcrypt_cbc (
string cipher, string key, string data, int mode [, string iv]) \linebreak
```

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_ciphertype 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 >= 4.0.0)

Encrypt/decrypt data in CFB mode

```
string mcrypt_cfb ( int cipher, string key, string data, int mode, string iv) \linebreak string mcrypt_cfb ( string
cipher, string key, string data, int mode [, string iv]) \linebreak
```

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_ciphertype 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_create_iv (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Create an initialization vector (IV) from a random source

string **mcrypt_create_iv** (int size, int source) \linebreak

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);
?>
```

The IV is only meant to give an alternative seed to the encryption routines. This IV does not need to be secret at all, though it can be desirable. You even can send it along with your ciphertext without losing security.

More information can be found at [???](#), [???](#) and in chapter 9.3 of Applied Cryptography by Schneier (ISBN 0-471-11709-9) for a discussion of this topic.

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]) \linebreak

mcrypt_decrypt() decrypts the data and returns the unencrypted data.

Cipher is one of the MCRYPT_ciphertype 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_ectb (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Encrypt/decrypt data in ECB mode

```
string mcrypt_ectb ( int cipher, string key, string data, int mode) \linebreak
string mcrypt_ectb ( string cipher, string key, string data, int mode [, string iv]) \linebreak
```

The first prototype is when linked against libmcrypt 2.2.x, the second when linked against libmcrypt 2.4.x.

mcrypt_ectb() 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_ciphernam 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_enc_get_algorithms_name (PHP 4 >= 4.0.2)

Returns the name of the opened algorithm

```
string mcrypt_enc_get_algorithms_name ( resource td) \linebreak
```

This function returns the name of the algorithm.

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) \linebreak
```

This function returns the block size of the algorithm specified by the encryption descriptor *td* in bytes.

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) \linebreak
```

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_key_size (PHP 4 >= 4.0.2)

Returns the maximum supported keysize of the opened mode

int **mcrypt_enc_get_key_size** (resource td) \linebreak

This function returns the maximum supported key size of the algorithm specified by the encryption descriptor td in bytes.

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) \linebreak

This function returns the name of the mode.

mcrypt_enc_get_supported_key_sizes (PHP 4 >= 4.0.2)

Returns an array with the supported key sizes of the opened algorithm

array **mcrypt_enc_get_supported_key_sizes** (resource td) \linebreak

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_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) \linebreak

This function returns 1 if the algorithm is a block algorithm, or 0 if it is a stream algorithm.

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) \linebreak
```

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_mode (PHP 4 >= 4.0.2)

Checks whether the opened mode outputs blocks

```
int mcrypt_enc_is_block_mode ( resource td) \linebreak
```

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_self_test (PHP 4 >= 4.0.2)

This function runs a self test on the opened module

```
int mcrypt_enc_self_test ( resource td) \linebreak
```

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_encrypt (PHP 4 >= 4.0.2)

Encrypts plaintext with given parameters

```
string mcrypt_encrypt ( string cipher, string key, string data, string mode [, string iv]) \linebreak
```

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_generic (PHP 4 >= 4.0.2)

This function encrypts data

string **mcrypt_generic** (resource *td*, string *data*) \linebreak

This function encrypts data. The data is padded with "\0" to make sure the length of the data is $n * \text{blocksize}$. This function returns the encrypted data. Note that the length of the returned string can in fact be longer than the input, due to the padding of the data.

mcrypt_generic_deinit (PHP 4 >= 4.1.1)

This function terminates encrypt specified by the descriptor *td*

bool **mcrypt_generic_deinit** (resource *td*) \linebreak

Warning

This function is currently not documented; only the argument list is available.

mcrypt_generic_end (PHP 4 >= 4.0.2)

This function terminates encryption

```
bool mcrypt_generic_end ( resource td) \linebreak
```

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 succes.

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) \linebreak
```

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 a negative value on error.

You need to call this function before every `mcrypt_generic()` or `mdecrypt_generic()`.

mcrypt_get_block_size (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Get the block size of the specified cipher

```
int mcrypt_get_block_size ( int cipher) \linebreak
int mcrypt_get_block_size ( string cipher, string module) \linebreak
```

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_cipher_name (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Get the name of the specified cipher

```
string mcrypt_get_cipher_name ( int cipher) \linebreak string mcrypt_get_cipher_name ( string cipher) \linebreak
```

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_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) \linebreak int mcrypt_get_iv_size ( resource td) \linebreak
```

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_ciphertype` 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_get_key_size (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Get the key size of the specified cipher

```
int mcrypt_get_key_size ( int cipher) \linebreak
int mcrypt_get_key_size ( string cipher, string module) \linebreak
```

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_list_algorithms (PHP 4 >= 4.0.2)

Get an array of all supported ciphers

```
array mcrypt_list_algorithms ( [string lib_dir]) \linebreak
```

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 specifies, 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]) \linebreak

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 specifies, 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_module_close (PHP 4 >= 4.0.2)

Free the descriptor td

bool **mcrypt_module_close** (resource td) \linebreak

Warning

This function is currently not documented; only the argument list is available.

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]) \linebreak
```

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]) \linebreak
```

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_supported_key_sizes (PHP 4 >= 4.0.2)

Returns an array with the supported key sizes of the opened algorithm

```
array mcrypt_module_get_supported_key_sizes ( string algorithm [, string lib_dir]) \linebreak
```

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.

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]) \linebreak
```

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_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]) \linebreak

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_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]) \linebreak

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 cfb and stream). The optional *lib_dir* parameter can contain the location where the mode module is on the system.

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) \linebreak

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 (usally /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 EBC mode from the directory `/usr/lib/mcrypt-modes`.

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]) \linebreak
```

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_ofb (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Encrypt/decrypt data in OFB mode

```
string mcrypt_ofb ( int cipher, string key, string data, int mode, string iv) \linebreak string mcrypt_ofb ( string  
cipher, string key, string data, int mode [, string iv]) \linebreak
```

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_ciphertype` 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()`.

mdecrypt_generic (PHP 4 >= 4.0.2)

This function decrypts data

```
string mdecrypt_generic ( resource td, string data) \linebreak
```

This function decrypts data. Note that the length of the returned string can in fact be longer than 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.

LVI. 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)."<br />\n";
$hash = mhash (MHASH_MD5, $input, "Jefe");
print "The hmac is ".bin2hex ($hash)."<br />\n";
?>
```

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 (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Compute hash

string **mhash** (int hash, string data [, string key]) \linebreak

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_count (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Get the highest available hash id

int **mhash_count** (void) \linebreak

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_get_block_size (PHP 3>= 3.0.9, PHP 4 >= 4.0.0)

Get the block size of the specified hash

int **mhash_get_block_size** (int hash) \linebreak

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_get_hash_name (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Get the name of the specified hash

string **mhash_get_hash_name** (int hash) \linebreak

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_keygen_s2k (PHP 4 >= 4.0.4)

Generates a key

string **mhash_keygen_s2k** (int hash, string password, string salt, int bytes) \linebreak

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.

LVII. Microsoft SQL Server functions

The MSSQL extension is available on Win32 systems only. You can use the Sybase extension to connect to MSSQL databases from other platforms.

These functions allow you to access MS SQL Server database. The extension requires the MS SQL Client Tools to be installed on the system where PHP is installed. The Client Tools can be installed from the MS SQL Server CD or by copying `ntwdblib.dll` from `\winnt\system32` on the server to `\winnt\system32` on the PHP box. Copying `ntwdblib.dll` will only provide access. Configuration of the client will require installation of all the tools.

The MSSQL extension is enabled by adding `extension=php_mssql.dll` to `php.ini`.

mssql_bind (PHP 4 >= 4.1.0)

Adds a parameter to a stored procedure or a remote stored procedure

```
int mssql_bind ( int stmt, string param_name, mixed var, int type [, int is_output [, int is_null [, int maxlen]])  
\linebreak
```

Warning

This function is currently not documented; only the argument list is available.

mssql_close (PHP 3, PHP 4 >= 4.0.0)

Close MS SQL Server connection

```
int mssql_close ( [int link_identifier]) \linebreak
```

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 >= 4.0.0)

Open MS SQL server connection

```
int mssql_connect ( [string servername [, string username [, string password]]) \linebreak
```

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 >= 4.0.0)

Move internal row pointer

```
int mssql_data_seek ( int result_identifier, int row_number) \linebreak
```

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_execute (PHP 4 >= 4.1.0)

Executes a stored procedure on a MS-SQL server database

```
int mssql_execute ( int stmt) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

mssql_fetch_array (PHP 3, PHP 4 >= 4.0.0)

Fetch row as array

```
int mssql_fetch_array ( int result) \linebreak
```

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_assoc (PHP 4 >= 4.2.0)

Returns an associative array of the current row in the result set specified by result_id

array **mssql_fetch_assoc** (int result_id [, int result_type]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

mssql_fetch_batch (PHP 4 >= 4.0.4)

Returns the next batch of records

int **mssql_fetch_batch** (string result_index) \linebreak

Warning

This function is currently not documented; only the argument list is available.

mssql_fetch_field (PHP 3, PHP 4 >= 4.0.0)

Get field information

object **mssql_fetch_field** (int result [, int field_offset]) \linebreak

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 >= 4.0.0)

Fetch row as object

```
int mssql_fetch_object ( int result) \linebreak
```

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 >= 4.0.0)

Get row as enumerated array

```
array mssql_fetch_row ( int result) \linebreak
```

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 >= 4.0.0)

Get the length of a field

```
int mssql_field_length ( int result [, int offset]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

mssql_field_name (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Get the name of a field

```
int mssql_field_name ( int result [, int offset]) \linebreak
```

mssql_field_seek (PHP 3, PHP 4 >= 4.0.0)

Set field offset

```
int mssql_field_seek ( int result, int field_offset) \linebreak
```

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 >= 4.0.0)

Get the type of a field

```
string mssql_field_type ( int result [, int offset]) \linebreak
```

mssql_free_result (PHP 3, PHP 4 >= 4.0.0)

Free result memory

```
int mssql_free_result ( int result) \linebreak
```

`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 >= 4.0.0)

Returns the last message from server (over `min_message_severity`?)

```
string mssql_get_last_message ( void) \linebreak
```


mssql_guid_string (PHP 4 >= 4.1.0)

Converts a 16 byte binary GUID to a string

string **mssql_guid_string** (string binary [, int short_format]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

mssql_init (PHP 4 >= 4.1.0)

Initializes a stored procedure or a remote stored procedure

int **mssql_init** (string sp_name [, int conn_id]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

mssql_min_error_severity (PHP 3, PHP 4 >= 4.0.0)

Sets the lower error severity

void **mssql_min_error_severity** (int severity) \linebreak

mssql_min_message_severity (PHP 3, PHP 4 >= 4.0.0)

Sets the lower message severity

void **mssql_min_message_severity** (int severity) \linebreak

mssql_next_result (PHP 4 >= 4.0.5)

Move the internal result pointer to the next result

bool **mssql_next_result** (int result_id) \linebreak

When sending more than one SQL statement to the server or executing a stored procedure with multiple results, it will cause the server to return multiple result sets. This function will test for additional results available from the server. If an additional result set exists it will free the existing result set and prepare to fetch the rows from the new result set. The function will return TRUE if an additional result set was available or FALSE otherwise.

Example 1. mssql_next_result() example

```
<?php
$link = mssql_connect ("localhost", "userid", "secret");
mssql_select_db("MyDB", $link);
$SQL = "Select * from table1 select * from table2";
$rs = mssql_query($SQL, $link);
do {
    while ($row = mssql_fetch_row($rs)) {
    }
} while (mssql_next_result($rs));
mssql_free_result($rs);
mssql_close ($link);
?>
```

mssql_num_fields (PHP 3, PHP 4 >= 4.0.0)

Get number of fields in result

int **mssql_num_fields** (int result) \linebreak

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 >= 4.0.0)

Get number of rows in result

int **mssql_num_rows** (string result) \linebreak

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 >= 4.0.0)

Open persistent MS SQL connection

```
int mssql_pconnect ( [string servername [, string username [, string password]]]) \linebreak
```

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 >= 4.0.0)

Send MS SQL query

```
int mssql_query ( string query [, int link_identifier]) \linebreak
```

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 >= 4.0.0)

Get result data

```
int mssql_result ( int result, int i, mixed field) \linebreak
```

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_rows_affected (PHP 4 >= 4.0.4)

Returns the number of records affected by the query

```
int mssql_rows_affected ( int conn_id) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

mssql_select_db (PHP 3, PHP 4 >= 4.0.0)

Select MS SQL database

```
int mssql_select_db ( string database_name [, int link_identifier]) \linebreak
```

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()`

LVIII. Ming functions for Flash

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

Introduction

Ming is an open-source (LGPL) library which allows you to create SWF ("Flash") format movies. Ming supports almost all of Flash 4's features, including: shapes, gradients, bitmaps (pngs and jpegs), morphs ("shape tweens"), text, buttons, actions, sprites ("movie clips"), streaming mp3, and color transforms--the only thing that's missing is sound events.

Ming is not an acronym.

Note that all values specifying length, distance, size, etc. are in "twips", twenty units per pixel. That's pretty much arbitrary, though, since the player scales the movie to whatever pixel size is specified in the embed/object tag, or the entire frame if not embedded.

Ming offers a number of advantages over the existing PHP/libswf module. You can use Ming anywhere you can compile the code, whereas libswf is closed-source and only available for a few platforms, Windows not one of them. Ming provides some insulation from the mundane details of the SWF file format, wrapping the movie elements in PHP objects. Also, Ming is still being maintained; if there's a feature that you want to see, just let us know ming@opaque.net (mailto:ming@opaque.net).

Ming was added in PHP 4.0.5.

Installation

To use Ming with PHP, you first need to build and install the Ming library. Source code and installation instructions are available at the Ming home page : <http://www.opaque.net/ming/> along with examples, a small tutorial, and the latest news.

Download the ming archive. Unpack the archive. Go in the Ming directory. `make`. `make install`.

This will build `libming.so` and install it into `/usr/lib/`, and copy `ming.h` into `/usr/include/`. Edit the `PREFIX=` line in the `Makefile` to change the installation directory.

built into php (unix)

```
mkdir <phpdir>/ext/ming
cp php_ext/* <phpdir>/ext/ming
```

```
cd <phpdir>
./buildconf
./configure --with-ming <other config options>
```

Build and install php as usual, Restart web server if necessary

built into php (unix)

download `php_ming.so.gz`. uncompress it and copy it to your php modules directory. (you can find your php module directory by running **php-config --extension-dir**). Now either just add `extension=php_ming.so` to your `php.ini` file, or put `dl('php_ming.so');` at the head of all of your Ming scripts.

How to use Ming

Ming introduces 13 new objects in PHP, all with matching methods and attributes. To use them, you need to know about objects.

- `swfmovie()`.
- `swfshape()`.
- `swfdisplayitem()`.
- `swfgradient()`.
- `swfbitmap()`.
- `swffill()`.
- `swfmorph()`.
- `swftext()`.
- `swffont()`.
- `swftextfield()`.
- `swfsprite()`.
- `swfbutton()`.
- `swfaction()`.

ming_setcubicthreshold (PHP 4 >= 4.0.5)

Set cubic threshold (?)

```
void ming_setcubicthreshold ( int threshold) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ming_setscale (PHP 4 >= 4.0.5)

Set scale (?)

```
void ming_setscale ( int scale) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

ming_useswfversion (PHP 4 >= 4.2.0)

Use SWF version (?)

```
void ming_useswfversion ( int version) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

SWFAction (PHP 4 >= 4.0.5)

Creates a new Action.

new **swfaction** (string script) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfaction() creates a new Action, and compiles the given script into an SWFAction object.

The script syntax is based on the C language, but with a lot taken out- the SWF bytecode machine is just too simpleminded to do a lot of things we might like. For instance, we can't implement function calls without a tremendous amount of hackery because the jump bytecode has a hardcoded offset value. No pushing your calling address to the stack and returning- every function would have to know exactly where to return to.

So what's left? The compiler recognises the following tokens:

- break
- for
- continue
- if
- else
- do
- while

There is no typed data; all values in the SWF action machine are stored as strings. The following functions can be used in expressions:

time()

Returns the number of milliseconds (?) elapsed since the movie started.

random(seed)

Returns a pseudo-random number in the range 0-seed.

length(expr)

Returns the length of the given expression.

int(number)

Returns the given number rounded down to the nearest integer.

concat(expr, expr)

Returns the concatenation of the given expressions.

ord(expr)

Returns the ASCII code for the given character

chr(num)

Returns the character for the given ASCII code

substr(string, location, length)

Returns the substring of length length at location location of the given string string.

Additionally, the following commands may be used:

duplicateClip(clip, name, depth)

Duplicate the named movie clip (aka sprite). The new movie clip has name name and is at depth depth.

removeClip(expr)

Removes the named movie clip.

trace(expr)

Write the given expression to the trace log. Doubtful that the browser plugin does anything with this.

startDrag(target, lock, [left, top, right, bottom])

Start dragging the movie clip target. The lock argument indicates whether to lock the mouse (?)-use 0 (FALSE) or 1 (TRUE). Optional parameters define a bounding area for the dragging.

stopDrag()

Stop dragging my heart around. And this movie clip, too.

callFrame(expr)

Call the named frame as a function.

getURL(url, target, [method])

Load the given URL into the named target. The target argument corresponds to HTML document targets (such as "_top" or "_blank"). The optional method argument can be POST or GET if you want to submit variables back to the server.

loadMovie(url, target)

Load the given URL into the named target. The target argument can be a frame name (I think), or one of the magical values "_level0" (replaces current movie) or "_level1" (loads new movie on top of current movie).

nextFrame()

Go to the next frame.

prevFrame()

Go to the last (or, rather, previous) frame.

play()

Start playing the movie.

stop()

Stop playing the movie.

toggleQuality()

Toggle between high and low quality.

stopSounds()

Stop playing all sounds.

gotoFrame(num)

Go to frame number num. Frame numbers start at 0.

gotoFrame(name)

Go to the frame named name. Which does a lot of good, since I haven't added frame labels yet.

setTarget(expr)

Sets the context for action. Or so they say- I really have no idea what this does.

And there's one weird extra thing. The expression `frameLoaded(num)` can be used in if statements and while loops to check if the given frame number has been loaded yet. Well, it's supposed to, anyway, but I've never tested it and I seriously doubt it actually works. You can just use `:/framesLoaded` instead.

Movie clips (all together now- aka sprites) have properties. You can read all of them (or can you?), you can set some of them, and here they are:

- x
- y
- xScale
- yScale
- currentFrame - (read-only)
- totalFrames - (read-only)
- alpha - transparency level
- visible - 1=on, 0=off (?)
- width - (read-only)
- height - (read-only)
- rotation
- target - (read-only) (???)

- framesLoaded - (read-only)
- name
- dropTarget - (read-only) (???)
- url - (read-only) (???)
- highQuality - 1=high, 0=low (?)
- focusRect - (???)
- soundBufTime - (???)

So, setting a sprite's x position is as simple as `/box.x = 100;`. Why the slash in front of the box, though? That's how flash keeps track of the sprites in the movie, just like a unix filesystem- here it shows that box is at the top level. If the sprite named box had another sprite named biff inside of it, you'd set its x position with `/box/biff.x = 100;`. At least, I think so; correct me if I'm wrong here.

This simple example will move the red square across the window.

Example 1. swfaction() example

```
<?php
$s = new SWFShape();
$f = $s->addFill(0xff, 0, 0);
$s->setRightFill($f);

$s->movePenTo(-500,-500);
$s->drawLineTo(500,-500);
$s->drawLineTo(500,500);
$s->drawLineTo(-500,500);
$s->drawLineTo(-500,-500);

$p = new SWFSprite();
$i = $p->add($s);
$i->setDepth(1);
$p->nextFrame();

for($n=0; $n<5; ++$n)
{
    $i->rotate(-15);
    $p->nextFrame();
}

$m = new SWFMovie();
$m->setBackground(0xff, 0xff, 0xff);
$m->setDimension(6000,4000);

$i = $m->add($p);
$i->setDepth(1);
$i->moveTo(-500,2000);
$i->setName("box");

$m->add(new SWFAction("/box.x += 3;"));
```

```

$m->nextFrame();
$m->add(new SWFAction("gotoFrame(0); play();"));
$m->nextFrame();

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

This simple example tracks down your mouse on the screen.

Example 2. swfaction() example

```

<?php

$m = new SWFMovie();
$m->setRate(36.0);
$m->setDimension(1200, 800);
$m->setBackground(0, 0, 0);

/* mouse tracking sprite - empty, but follows mouse so we can
   get its x and y coordinates */

$i = $m->add(new SWFSprite());
$i->setName('mouse');

$m->add(new SWFAction("
    startDrag('/mouse', 1); /* '1' means lock sprite to the mouse */
"));

/* might as well turn off antialiasing, since these are just squares. */

$m->add(new SWFAction("
    this.quality = 0;
"));

/* morphing box */
$r = new SWFMorph();
$s = $r->getShapel();

/* Note this is backwards from normal shapes. No idea why. */
$s->setLeftFill($s->addFill(0xff, 0xff, 0xff));
$s->movePenTo(-40, -40);
$s->drawLine(80, 0);
$s->drawLine(0, 80);
$s->drawLine(-80, 0);
$s->drawLine(0, -80);

$s = $r->getShape2();

```

```

$s->setLeftFill($s->addFill(0x00, 0x00, 0x00));
$s->movePenTo(-1, -1);
$s->drawLine(2, 0);
$s->drawLine(0, 2);
$s->drawLine(-2, 0);
$s->drawLine(0, -2);

/* sprite container for morphing box -
   this is just a timeline w/ the box morphing */

$box = new SWFSprite();
$box->add(new SWFAction("
    stop();
"));
$i = $box->add($r);

for($n=0; $n<=20; ++$n)
{
    $i->setRatio($n/20);
    $box->nextFrame();
}

/* this container sprite allows us to use the same action code many times */

$cell = new SWFSprite();
$i = $cell->add($box);
$i->setName('box');

$cell->add(new SWFAction("

    setTarget('box');

    /* ...x means the x coordinate of the parent, i.e. (...)x */
    dx = (/mouse.x + random(6)-3 - ...x)/5;
    dy = (/mouse.y + random(6)-3 - ...y)/5;
    gotoFrame(int(dx*dx + dy*dy));

"));

$cell->nextFrame();
$cell->add(new SWFAction("

    gotoFrame(0);
    play();

"));

$cell->nextFrame();

/* finally, add a bunch of the cells to the movie */

for($x=0; $x<12; ++$x)

```

```

{
  for($y=0; $y<8; ++$y)
  {
    $i = $m->add($cell);
    $i->moveTo(100*$x+50, 100*$y+50);
  }
}

$m->nextFrame();

$m->add(new SWFAction("

  gotoFrame(1);
  play();

"));

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

Same as above, but with nice colored balls...

Example 3. swfaction() example

```

<?php

$m = new SWFMovie();
$m->setDimension(11000, 8000);
$m->setBackground(0x00, 0x00, 0x00);

$m->add(new SWFAction("

this.quality = 0;
/frames.visible = 0;
startDrag('/mouse', 1);

"));

// mouse tracking sprite
$t = new SWFSprite();
$i = $m->add($t);
$i->setName('mouse');

$g = new SWFGradient();
$g->addEntry(0, 0xff, 0xff, 0xff, 0xff);
$g->addEntry(0.1, 0xff, 0xff, 0xff, 0xff);
$g->addEntry(0.5, 0xff, 0xff, 0xff, 0x5f);

```

```

$g->addEntry(1.0, 0xff, 0xff, 0xff, 0);

// gradient shape thing
$s = new SWFShape();
$f = $s->addFill($g, SWFFILL_RADIAL_GRADIENT);
$f->scaleTo(0.03);
$s->setRightFill($f);
$s->movePenTo(-600, -600);
$s->drawLine(1200, 0);
$s->drawLine(0, 1200);
$s->drawLine(-1200, 0);
$s->drawLine(0, -1200);

// need to make this a sprite so we can multColor it
$p = new SWFSprite();
$p->add($s);
$p->nextFrame();

// put the shape in here, each frame a different color
$q = new SWFSprite();
$q->add(new SWFAction("gotoFrame(random(7)+1); stop();"));
$i = $q->add($p);

$i->multColor(1.0, 1.0, 1.0);
$q->nextFrame();
$i->multColor(1.0, 0.5, 0.5);
$q->nextFrame();
$i->multColor(1.0, 0.75, 0.5);
$q->nextFrame();
$i->multColor(1.0, 1.0, 0.5);
$q->nextFrame();
$i->multColor(0.5, 1.0, 0.5);
$q->nextFrame();
$i->multColor(0.5, 0.5, 1.0);
$q->nextFrame();
$i->multColor(1.0, 0.5, 1.0);
$q->nextFrame();

// finally, this one contains the action code
$p = new SWFSprite();
$i = $p->add($q);
$i->setName('frames');
$p->add(new SWFAction("

dx = (/:mousex-/:lastx)/3 + random(10)-5;
dy = (/:mousey-/:lasty)/3;
x = /:mousex;
y = /:mousey;
alpha = 100;

"));
$p->nextFrame();

```

```

    $p->add(new SWFAction("

this.x = x;
this.y = y;
this.alpha = alpha;
x += dx;
y += dy;
dy += 3;
alpha -= 8;

    ));
    $p->nextFrame();

    $p->add(new SWFAction("prevFrame(); play();"));
    $p->nextFrame();

    $i = $m->add($p);
    $i->setName('frames');
    $m->nextFrame();

    $m->add(new SWFAction("

lastx = mousex;
lasty = mousey;
mousex = /mouse.x;
mousey = /mouse.y;

++num;

if(num == 11)
    num = 1;

removeClip('char' & num);
duplicateClip(/frames, 'char' & num, num);

    ));

    $m->nextFrame();
    $m->add(new SWFAction("prevFrame(); play();"));

    header('Content-type: application/x-shockwave-flash');
    $m->output();
?>

```

This simple example will handles keyboard actions. (You'll probably have to click in the window to give it focus. And you'll probably have to leave your mouse in the frame, too. If you know how to give buttons focus programatically, feel free to share, won't you?)

Example 4. swfaction() example

```

<?php

    /* sprite has one letter per frame */

    $p = new SWFSprite();
    $p->add(new SWFAction("stop();"));

    $chars = "abcdefghijklmnopqrstuvwxy".
             "ABCDEFGHIJKLMNPOQRSTUVWXYZ".
             "1234567890!@#$%^&/*()_+--=/[ ]{|};:,.<>¿~";

    $f = new SWFFont("_sans");

    for($n=0; $nremove($i);
        $t = new SWFTextField();
        $t->setFont($f);
        $t->setHeight(240);
        $t->setBounds(600,240);
        $t->align(SWFTEXTFIELD_ALIGN_CENTER);
        $t->addString($c);
        $i = $p->add($t);
        $p->labelFrame($c);
        $p->nextFrame();
    }

    /* hit region for button - the entire frame */

    $s = new SWFShape();
    $s->setFillStyle0($s->addSolidFill(0, 0, 0, 0));
    $s->drawLine(600, 0);
    $s->drawLine(0, 400);
    $s->drawLine(-600, 0);
    $s->drawLine(0, -400);

    /* button checks for pressed key, sends sprite to the right frame */

    $b = new SWFButton();
    $b->addShape($s, SWFBUTTON_HIT);

    for($n=0; $naddAction(new SWFAction("

setTarget('/char');
gotoFrame('$c');

        "), SWFBUTTON_KEYPRESS($c));
    }

    $m = new SWFMovie();
    $m->setDimension(600,400);
    $i = $m->add($p);

```

```

$i->setName('char');
$i->moveTo(0,80);

$m->add($b);

header('Content-type: application/x-shockwave-flash');
$m->output();

?>

```

SWFBitmap (PHP 4 >= 4.0.5)

Loads Bitmap object

`new swfbitmap (string filename [, int alphafilename]) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbitmap() creates a new SWFBitmap object from the Jpeg or DBL file named *filename*. *alphafilename* indicates a MSK file to be used as an alpha mask for a Jpeg image.

Note: We can only deal with baseline (frame 0) jpegs, no baseline optimized or progressive scan jpegs!

SWFBitmap has the following methods : `swfbitmap->getwidth()` and `swfbitmap->getheight()`.

You can't import png images directly, though- have to use the png2dbl utility to make a dbl ("define bits lossless") file from the png. The reason for this is that I don't want a dependency on the png library in ming- autoconf should solve this, but that's not set up yet.

Example 1. Import PNG files

```

<?php
$s = new SWFShape();
$f = $s->addFill(new SWFBitmap("png.dbl"));
$s->setRightFill($f);

$s->drawLine(32, 0);
$s->drawLine(0, 32);

```

```

$s->drawLine(-32, 0);
$s->drawLine(0, -32);

$m = new SWFMovie();
$m->setDimension(32, 32);
$m->add($s);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

And you can put an alpha mask on a jpeg fill.

Example 2. swfbitmap() example

```

<?php

$s = new SWFShape();

// .msk file generated with "gif2mask" utility
$f = $s->addFill(new SWFBitmap("alphafill.jpg", "alphafill.msk"));
$s->setRightFill($f);

$s->drawLine(640, 0);
$s->drawLine(0, 480);
$s->drawLine(-640, 0);
$s->drawLine(0, -480);

$c = new SWFShape();
$c->setRightFill($c->addFill(0x99, 0x99, 0x99));
$c->drawLine(40, 0);
$c->drawLine(0, 40);
$c->drawLine(-40, 0);
$c->drawLine(0, -40);

$m = new SWFMovie();
$m->setDimension(640, 480);
$m->setBackground(0xcc, 0xcc, 0xcc);

// draw checkerboard background
for($y=0; $y<480; $y+=40)
{
    for($x=0; $x<640; $x+=80)
    {
        $i = $m->add($c);
        $i->moveTo($x, $y);
    }
}

```

```

    $y+=40;

    for($x=40; $x<640; $x+=80)
    {
        $i = $m->add($c);
        $i->moveTo($x, $y);
    }
}

$m->add($s);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFBitmap->getHeight (unknown)

Returns the bitmap's height.

```
int swfbitmap->getheight ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbitmap->getheight() returns the bitmap's height in pixels.

See also **swfbitmap->getwidth()**.

SWFBitmap->getWidth (unknown)

Returns the bitmap's width.

```
int swfbitmap->getwidth ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbitmap->getwidth() returns the bitmap's width in pixels.

See also **swfbitmap->getheight()**.

SWFbutton (PHP 4 >= 4.0.5)

Creates a new Button.

new **swfbutton** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton() creates a new Button. Roll over it, click it, see it call action code. Swank.

SWFButton has the following methods : **swfbutton->addshape()**, **swfbutton->setup()**, **swfbutton->setover()** **swfbutton->setdown()**, **swfbutton->sethit()** **swfbutton->setaction()** and **swfbutton->addaction()**.

This simple example will show your usual interactions with buttons : rollover, rollon, mouseup, mousedown, noaction.

Example 1. swfbutton() example

```
<?php

    $f = new SWFFont("_serif");

    $p = new SWFSprite();

    function label($string)
    {
        global $f;

        $t = new SWFTextField();
        $t->setFont($f);
        $t->addString($string);
        $t->setHeight(200);
        $t->setBounds(3200,200);
        return $t;
    }
    function addLabel($string)
    {
        global $p;

        $i = $p->add(label($string));
```

```

    $p->nextFrame();
    $p->remove($i);
}

$p->add(new SWFAction("stop();"));
addLabel("NO ACTION");
addLabel("SWFBUTTON_MOUSEUP");
addLabel("SWFBUTTON_MOUSEDOWN");
addLabel("SWFBUTTON_MOUSEOVER");
addLabel("SWFBUTTON_MOUSEOUT");
addLabel("SWFBUTTON_MOUSEUPOUTSIDE");
addLabel("SWFBUTTON_DRAGOVER");
addLabel("SWFBUTTON_DRAGOUT");

function rect($r, $g, $b)
{
    $s = new SWFShape();
    $s->setRightFill($s->addFill($r, $g, $b));
    $s->drawLine(600,0);
    $s->drawLine(0,600);
    $s->drawLine(-600,0);
    $s->drawLine(0,-600);

    return $s;
}

$b = new SWFButton();
$b->addShape(rect(0xff, 0, 0), SWFBUTTON_UP | SWFBUTTON_HIT);
$b->addShape(rect(0, 0xff, 0), SWFBUTTON_OVER);
$b->addShape(rect(0, 0, 0xff), SWFBUTTON_DOWN);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(1);"),
    SWFBUTTON_MOUSEUP);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(2);"),
    SWFBUTTON_MOUSEDOWN);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(3);"),
    SWFBUTTON_MOUSEOVER);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(4);"),
    SWFBUTTON_MOUSEOUT);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(5);"),
    SWFBUTTON_MOUSEUPOUTSIDE);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(6);"),
    SWFBUTTON_DRAGOVER);

$b->addAction(new SWFAction("setTarget('/label'); gotoFrame(7);"),
    SWFBUTTON_DRAGOUT);

$m = new SWFMovie();

```

```

$m->setDimension(4000,3000);

$i = $m->add($p);
$i->setName("label");
$i->moveTo(400,1900);

$i = $m->add($b);
$i->moveTo(400,900);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

This simple example will enables you to drag draw a big red button on the windows. No drag-and-drop, just moving around.

Example 2. swfbutton->addaction() example

```

<?php

$s = new SWFShape();
$s->setRightFill($s->addFill(0xff, 0, 0));
$s->drawLine(1000,0);
$s->drawLine(0,1000);
$s->drawLine(-1000,0);
$s->drawLine(0,-1000);

$b = new SWFButton();
$b->addShape($s, SWFBUTTON_HIT | SWFBUTTON_UP | SWFBUTTON_DOWN | SWFBUTTON_OVER);

$b->addAction(new SWFAction("startDrag('/test', 0);"), // '0' means don't lock to mouse
              SWFBUTTON_MOUSEDOWN);

$b->addAction(new SWFAction("stopDrag();"),
              SWFBUTTON_MOUSEUP | SWFBUTTON_MOUSEUPOUTSIDE);

$p = new SWFSprite();
$p->add($b);
$p->nextFrame();

$m = new SWFMovie();
$i = $m->add($p);
$i->setName('test');
$i->moveTo(1000,1000);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

swfbutton_keypress (PHP 4 >= 4.0.5)

Returns the action flag for keyPress(char)

```
int swfbutton_keypress ( string str) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

SWFbutton->addAction (unknown)

Adds an action

```
void swfbutton->addaction ( ressource action, int flags) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->addaction() adds the action *action* to this button for the given conditions. The following *flags* are valid: SWFBUTTON_MOUSEOVER, SWFBUTTON_MOUSEOUT, SWFBUTTON_MOUSEUP, SWFBUTTON_MOUSEUPOUTSIDE, SWFBUTTON_MOUSEDOWN, SWFBUTTON_DRAGOUT and SWFBUTTON_DRAGOVER.

See also **swfbutton->addshape()** and **SWFAction()**.

SWFbutton->addShape (unknown)

Adds a shape to a button

```
void swfbutton->addshape ( ressource shape, int flags) \linebreak
```


Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->addshape() adds the shape *shape* to this button. The following *flags*' values are valid: SWFBUTTON_UP, SWFBUTTON_OVER, SWFBUTTON_DOWN or SWFBUTTON_HIT. SWFBUTTON_HIT isn't ever displayed, it defines the hit region for the button. That is, everywhere the hit shape would be drawn is considered a "touchable" part of the button.

SWFbutton->setAction (unknown)

Sets the action

```
void swfbutton->setaction ( resource action) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->setaction() sets the action to be performed when the button is clicked. Alias for `addAction(shape, SWFBUTTON_MOUSEUP)`. *action* is a `swfaction()`.

See also **swfbutton->addshape()** and **SWFAction()**.

SWFbutton->setdown (unknown)

Alias for `addShape(shape, SWFBUTTON_DOWN)`

```
void swfbutton->setdown ( resource shape) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->setdown() alias for `addShape(shape, SWFBUTTON_DOWN)`.

See also **swfbutton->addshape()** and **SWFAction()**.

SWFbutton->setHit (unknown)

Alias for addShape(shape, SWFBUTTON_HIT)

void **swfbutton->sethit** (ressource shape) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->sethit() alias for addShape(shape, SWFBUTTON_HIT).

See also **swfbutton->addshape**() and **SWFAction**().

SWFbutton->setOver (unknown)

Alias for addShape(shape, SWFBUTTON_OVER)

void **swfbutton->setover** (ressource shape) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->setover() alias for addShape(shape, SWFBUTTON_OVER).

See also **swfbutton->addshape**() and **SWFAction**().

SWFbutton->setUp (unknown)

Alias for addShape(shape, SWFBUTTON_UP)

void **swfbutton->setup** (ressource shape) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfbutton->setup() alias for addShape(shape, SWFBUTTON_UP).

See also `swfbutton->addshape()` and `SWFAction()`.

SWFDisplayItem (unknown)

Creates a new displayitem object.

```
new swfdisplayitem ( void ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfdisplayitem()` creates a new `swfdisplayitem` object.

Here's where all the animation takes place. After you define a shape, a text object, a sprite, or a button, you add it to the movie, then use the returned handle to move, rotate, scale, or skew the thing.

SWFDisplayItem has the following methods : `swfdisplayitem->move()`, `swfdisplayitem->moveto()`, `swfdisplayitem->scaletto()`, `swfdisplayitem->scale()`, `swfdisplayitem->rotate()`, `swfdisplayitem->rotateto()`, `swfdisplayitem->skewxto()`, `swfdisplayitem->skewx()`, `swfdisplayitem->skewyto()`, `swfdisplayitem->skewyto()`, `swfdisplayitem->setdepth()`, `swfdisplayitem->remove()`, `swfdisplayitem->setname()`, `swfdisplayitem->setratio()`, `swfdisplayitem->addcolor()` and `swfdisplayitem->multicolor()`.

SWFDisplayItem->addColor (unknown)

Adds the given color to this item's color transform.

```
void swfdisplayitem->addcolor ( [int red [, int green [, int blue [, int a]]]] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfdisplayitem->addcolor()` adds the color to this item's color transform. The color is given in its RGB form.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

SWFDisplayItem->move (unknown)

Moves object in relative coordinates.

```
void swfdisplayitem->move ( int dx, int dy) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->move() moves the current object by (dx,dy) from its current position.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

See also `swfdisplayitem->moveto()`.

SWFDisplayItem->moveTo (unknown)

Moves object in global coordinates.

```
void swfdisplayitem->moveto ( int x, int y) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->moveto() moves the current object to (x,y) in global coordinates.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

See also `swfdisplayitem->move()`.

SWFDisplayItem->multColor (unknown)

Multiplies the item's color transform.

```
void swfdisplayitem->multicolor ( [int red [, int green [, int blue [, int a]]]]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->multicolor() multiplies the item's color transform by the given values.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

This simple example will modify your picture's atmosphere to Halloween (use a landscape or bright picture).

Example 1. swfdisplayitem->multicolor() example

```
<?php

    $b = new SWFBitmap("backyard.jpg");
    // note use your own picture :-
    $s = new SWFShape();
    $s->setRightFill($s->addFill($b));
    $s->drawLine($b->getWidth(), 0);
    $s->drawLine(0, $b->getHeight());
    $s->drawLine(-$b->getWidth(), 0);
    $s->drawLine(0, -$b->getHeight());

    $m = new SWFMovie();
    $m->setDimension($b->getWidth(), $b->getHeight());

    $i = $m->add($s);

    for($n=0; $n<=20; ++$n)
    {
        $i->multColor(1.0-$n/10, 1.0, 1.0);
        $i->addColor(0xff*$n/20, 0, 0);
        $m->nextFrame();
    }

    header('Content-type: application/x-shockwave-flash');
    $m->output();
?>
```

SWFDisplayItem->remove (unknown)

Removes the object from the movie

```
void swfdisplayitem->remove ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->remove() removes this object from the movie's display list.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the **swfmovie->add()**.

See also **swfmovie->add()**.

SWFDisplayItem->Rotate (unknown)

Rotates in relative coordinates.

```
void swfdisplayitem->rotate ( float ddegrees) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->rotate() rotates the current object by *ddegrees* degrees from its current rotation.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the **swfmovie->add()**.

See also **swfdisplayitem->rotateto()**.

SWFDisplayItem->rotateTo (unknown)

Rotates the object in global coordinates.

```
void swfdisplayitem->rotateto ( float degrees) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->rotateto() set the current object rotation to *degrees* degrees in global coordinates.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

This example bring three rotating string from the background to the foreground. Pretty nice.

Example 1. swfdisplayitem->rotateto() example

```
<?php
    $thetext = "ming!";

    $f = new SWFFont("Bauhaus 93.fdb");

    $m = new SWFMovie();
    $m->setRate(24.0);
    $m->setDimension(2400, 1600);
    $m->setBackground(0xff, 0xff, 0xff);

    // functions with huge numbers of arbitrary
    // arguments are always a good idea! Really!

function text($r, $g, $b, $a, $rot, $x, $y, $scale, $string)
{
    global $f, $m;

    $t = new SWFText();
    $t->setFont($f);
    $t->setColor($r, $g, $b, $a);
    $t->setHeight(960);
    $t->moveTo(-($f->getWidth($string))/2, $f->getAscent()/2);
    $t->addString($string);

    // we can add properties just like a normal php var,
    // as long as the names aren't already used.
    // e.g., we can't set $i->scale, because that's a function

    $i = $m->add($t);
    $i->x = $x;
    $i->y = $y;
    $i->rot = $rot;
    $i->s = $scale;
    $i->rotateTo($rot);
    $i->scale($scale, $scale);

    // but the changes are local to the function, so we have to
```

```

    // return the changed object.  kinda weird..

    return $i;
}

function step($i)
{
    $oldrot = $i->rot;
    $i->rot = 19*$i->rot/20;
    $i->x = (19*$i->x + 1200)/20;
    $i->y = (19*$i->y + 800)/20;
    $i->s = (19*$i->s + 1.0)/20;

    $i->rotateTo($i->rot);
    $i->scaleTo($i->s, $i->s);
    $i->moveTo($i->x, $i->y);

    return $i;
}

// see?  it sure paid off in legibility:

$i1 = text(0xff, 0x33, 0x33, 0xff, 900, 1200, 800, 0.03, $thetext);
$i2 = text(0x00, 0x33, 0xff, 0x7f, -560, 1200, 800, 0.04, $thetext);
$i3 = text(0xff, 0xff, 0xff, 0x9f, 180, 1200, 800, 0.001, $thetext);

for($i=1; $i<=100; ++$i)
{
    $i1 = step($i1);
    $i2 = step($i2);
    $i3 = step($i3);

    $m->nextFrame();
}

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

See also `swfdisplayitem->rotate()`.

SWFDisplayItem->scale (unknown)

Scales the object in relative coordinates.

`void swfdisplayitem->scale (int dx, int dy) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->scale() scales the current object by (dx,dy) from its current size.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

See also **swfdisplayitem->scaletto()**.

SWFDisplayItem->scaleTo (unknown)

Scales the object in global coordinates.

void **swfdisplayitem->scaletto** (int x, int y) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->scaletto() scales the current object to (x,y) in global coordinates.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

See also **swfdisplayitem->scale()**.

SWFDisplayItem->setDepth (unknown)

Sets z-order

void **swfdisplayitem->setdepth** (float depth) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->rotate() sets the object's z-order to *depth*. Depth defaults to the order in which instances are created (by add'ing a shape/text to a movie)- newer ones are on top of older ones. If two objects are given the same depth, only the later-defined one can be moved.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

SWFDisplayItem->setName (unknown)

Sets the object's name

```
void swfdisplayitem->setname ( string name) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->setname() sets the object's name to *name*, for targeting with action script. Only useful on sprites.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

SWFDisplayItem->setRatio (unknown)

Sets the object's ratio.

```
void swfdisplayitem->setratio ( float ratio) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->setratio() sets the object's ratio to *ratio*. Obviously only useful for morphs.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

This simple example will morph nicely three concentric circles.

Example 1. swfdisplayitem->setname() example

```

<?php

    $p = new SWFMorph();

    $g = new SWFGradient();
    $g->addEntry(0.0, 0, 0, 0);
    $g->addEntry(0.16, 0xff, 0xff, 0xff);
    $g->addEntry(0.32, 0, 0, 0);
    $g->addEntry(0.48, 0xff, 0xff, 0xff);
    $g->addEntry(0.64, 0, 0, 0);
    $g->addEntry(0.80, 0xff, 0xff, 0xff);
    $g->addEntry(1.00, 0, 0, 0);

    $s = $p->getShapel();
    $f = $s->addFill($g, SWFFILL_RADIAL_GRADIENT);
    $f->scaleTo(0.05);
    $s->setLeftFill($f);
    $s->movePenTo(-160, -120);
    $s->drawLine(320, 0);
    $s->drawLine(0, 240);
    $s->drawLine(-320, 0);
    $s->drawLine(0, -240);

    $g = new SWFGradient();
    $g->addEntry(0.0, 0, 0, 0);
    $g->addEntry(0.16, 0xff, 0, 0);
    $g->addEntry(0.32, 0, 0, 0);
    $g->addEntry(0.48, 0, 0xff, 0);
    $g->addEntry(0.64, 0, 0, 0);
    $g->addEntry(0.80, 0, 0, 0xff);
    $g->addEntry(1.00, 0, 0, 0);

    $s = $p->getShape2();
    $f = $s->addFill($g, SWFFILL_RADIAL_GRADIENT);
    $f->scaleTo(0.05);
    $f->skewXTo(1.0);
    $s->setLeftFill($f);
    $s->movePenTo(-160, -120);
    $s->drawLine(320, 0);
    $s->drawLine(0, 240);
    $s->drawLine(-320, 0);
    $s->drawLine(0, -240);

    $m = new SWFMovie();
    $m->setDimension(320, 240);
    $i = $m->add($p);
    $i->moveTo(160, 120);

    for($n=0; $n<=1.001; $n+=0.01)
    {

```

```

    $i->setRatio($n);
    $m->nextFrame();
}

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFDisplayItem->skewX (unknown)

Sets the X-skew.

```
void swfdisplayitem->skewx ( float ddegrees ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->skewx() adds *ddegrees* to current x-skew.

The object may be a swfshape(), a swfbutton(), a swftext() or a swfsprite() object. It must have been added using the **swfmovie->add()**.

See also **swfdisplayitem->skewx()**, **swfdisplayitem->skewy()** and **swfdisplayitem->skewyto()**.

SWFDisplayItem->skewXTo (unknown)

Sets the X-skew.

```
void swfdisplayitem->skewxto ( float degrees ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfdisplayitem->skewxto() sets the x-skew to *degrees*. For *degrees* is 1.0, it means a 45-degree forward slant. More is more forward, less is more backward.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

See also `swfdisplayitem->skewx()`, `swfdisplayitem->skewy()` and `swfdisplayitem->skewyto()`.

SWFDisplayItem->skewY (unknown)

Sets the Y-skew.

```
void swfdisplayitem->skewy ( float ddegrees) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfdisplayitem->skewy()` adds *ddegrees* to current y-skew.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

See also `swfdisplayitem->skewyto()`, `swfdisplayitem->skewx()` and `swfdisplayitem->skewxto()`.

SWFDisplayItem->skewYTo (unknown)

Sets the Y-skew.

```
void swfdisplayitem->skewyto ( float degrees) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfdisplayitem->skewyto()` sets the y-skew to *degrees*. For *degrees* is 1.0, it means a 45-degree forward slant. More is more upward, less is more downward.

The object may be a `swfshape()`, a `swfbutton()`, a `swftext()` or a `swfsprite()` object. It must have been added using the `swfmovie->add()`.

See also `swfdisplayitem->skewy()`, `swfdisplayitem->skewx()` and `swfdisplayitem->skewxto()`.

SWFFill (PHP 4 >= 4.0.5)

Loads SWFFill object

The **swffill()** object allows you to transform (scale, skew, rotate) bitmap and gradient fills. **swffill()** objects are created by the **swfshape->addfill()** methods.

SWFFill has the following methods : **swffill->moveto()** and **swffill->scaletto()**, **swffill->rotateto()**, **swffill->skewxto()** and **swffill->skewyto()**.

SWFFill->moveTo (unknown)

Moves fill origin

```
void swffill->moveto ( int x, int y) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swffill->moveto() moves fill's origin to (x,y) in global coordinates.

SWFFill->rotateTo (unknown)

Sets fill's rotation

```
void swffill->rotateto ( float degrees) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swffill->rotateto() sets fill's rotation to *degrees* degrees.

SWFFill->scaleTo (unknown)

Sets fill's scale

void **swffill->scalet**o (int x, int y) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swffill->scaleto() sets fill's scale to x in the x-direction, y in the y-direction.

SWFFill->skewXTo (unknown)

Sets fill x-skew

void **swffill->skewx**to (float x) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swffill->skewxto() sets fill x-skew to x . For x is 1.0, it is a 45-degree forward slant. More is more forward, less is more backward.

SWFFill->skewYTo (unknown)

Sets fill y-skew

void **swffill->skewy**to (float y) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swffill->skewyto() sets fill y-skew to y . For y is 1.0, it is a 45-degree upward slant. More is more upward, less is more downward.

SWFFont (PHP 4 >= 4.0.5)

Loads a font definition

new **swffont** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

If *filename* is the name of an FDB file (i.e., it ends in ".fdb"), load the font definition found in said file. Otherwise, create a browser-defined font reference.

FDB ("font definition block") is a very simple wrapper for the SWF DefineFont2 block which contains a full description of a font. One may create FDB files from SWT Generator template files with the included makefdb utility- look in the util directory off the main ming distribution directory.

Browser-defined fonts don't contain any information about the font other than its name. It is assumed that the font definition will be provided by the movie player. The fonts `_serif`, `_sans`, and `_typewriter` should always be available. For example:

```
<?php
$f = newSWFFont( "_sans" );
?>
```

will give you the standard sans-serif font, probably the same as what you'd get with `` in HTML.

swffont() returns a reference to the font definition, for use in the **SWFText->setFont()** and the **SWFTextField->setFont()** methods.

SWFFont has the following methods : **swffont->getwidth()**.

swffont->getwidth (unknown)

Returns the string's width

int **swffont->getwidth** (string string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swffont->getwidth()` returns the string *string*'s width, using font's default scaling. You'll probably want to use the `SWFText()` version of this method which uses the text object's scale.

SWFGradient (PHP 4 >= 4.0.5)

Creates a gradient object

`new swfgradient (void) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfgradient()` creates a new SWFGradient object.

After you've added the entries to your gradient, you can use the gradient in a shape fill with the `swfshape->addfill()` method.

SWFGradient has the following methods : `swfgradient->addentry()`.

This simple example will draw a big black-to-white gradient as background, and a redish disc in its center.

Example 1. swfgradient() example

```
<?php

$m = new SWFMovie();
$m->setDimension(320, 240);

$s = new SWFShape();

// first gradient- black to white
$g = new SWFGradient();
$g->addEntry(0.0, 0, 0, 0);
$g->addEntry(1.0, 0xff, 0xff, 0xff);

$f = $s->addFill($g, SWFFILL_LINEAR_GRADIENT);
$f->scaleTo(0.01);
$f->moveTo(160, 120);
$s->setRightFill($f);
$s->drawLine(320, 0);
$s->drawLine(0, 240);
$s->drawLine(-320, 0);
$s->drawLine(0, -240);

$m->add($s);
```

```

$s = new SWFShape();

// second gradient- radial gradient from red to transparent
$g = new SWFGradient();
$g->addEntry(0.0, 0xff, 0, 0, 0xff);
$g->addEntry(1.0, 0xff, 0, 0, 0);

$f = $s->addFill($g, SWFFILL_RADIAL_GRADIENT);
$f->scaleTo(0.005);
$f->moveTo(160, 120);
$s->setRightFill($f);
$s->drawLine(320, 0);
$s->drawLine(0, 240);
$s->drawLine(-320, 0);
$s->drawLine(0, -240);

$m->add($s);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFGradient->addEntry (unknown)

Adds an entry to the gradient list.

```
void swfgradient->addentry ( float ratio, int red, int green, int blue [, int a] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfgradient->addentry() adds an entry to the gradient list. *ratio* is a number between 0 and 1 indicating where in the gradient this color appears. Thou shalt add entries in order of increasing ratio.

red, green, blue is a color (RGB mode). Last parameter *a* is optional.

SWFMorph (PHP 4 >= 4.0.5)

Creates a new SWFMorph object.

new **swfmorph** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmorph() creates a new SWFMorph object.

Also called a "shape tween". This thing lets you make those tacky twisting things that make your computer choke. Oh, joy!

The methods here are sort of weird. It would make more sense to just have newSWFMorph(shape1, shape2);, but as things are now, shape2 needs to know that it's the second part of a morph. (This, because it starts writing its output as soon as it gets drawing commands- if it kept its own description of its shapes and wrote on completion this and some other things would be much easier.)

SWFMorph has the following methods : **swfmorph->getshape1()** and **swfmorph->getshape1()**.

This simple example will morph a big red square into a smaller blue black-bordered square.

Example 1. swfmorph() example

```
<?php
    $p = new SWFMorph();

    $s = $p->getShape1();
    $s->setLine(0,0,0,0);

    /* Note that this is backwards from normal shapes (left instead of right).
       I have no idea why, but this seems to work.. */

    $s->setLeftFill($s->addFill(0xff, 0, 0));
    $s->movePenTo(-1000,-1000);
    $s->drawLine(2000,0);
    $s->drawLine(0,2000);
    $s->drawLine(-2000,0);
    $s->drawLine(0,-2000);

    $s = $p->getShape2();
    $s->setLine(60,0,0,0);
    $s->setLeftFill($s->addFill(0, 0, 0xff));
    $s->movePenTo(0,-1000);
    $s->drawLine(1000,1000);
    $s->drawLine(-1000,1000);
    $s->drawLine(-1000,-1000);
    $s->drawLine(1000,-1000);

    $m = new SWFMovie();
    $m->setDimension(3000,2000);
    $m->setBackground(0xff, 0xff, 0xff);
```

```

$i = $m->add($p);
$i->moveTo(1500,1000);

for($r=0.0; $r<=1.0; $r+=0.1)
{
    $i->setRatio($r);
    $m->nextFrame();
}

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFMorph->getshape1 (unknown)

Gets a handle to the starting shape

mixed **swfmorph->getshape1** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmorph->getshape1() gets a handle to the morph's starting shape. **swfmorph->getshape1()** returns an swfshape() object.

SWFMorph->getshape2 (unknown)

Gets a handle to the ending shape

mixed **swfmorph->getshape2** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmorph->getshape2() gets a handle to the morph's ending shape. **swfmorph->getshape2()** returns an **swfshape()** object.

SWFMovie (PHP 4 >= 4.0.5)

Creates a new movie object, representing an SWF version 4 movie.

new **swfmovie** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie() creates a new movie object, representing an SWF version 4 movie.

SWFMovie has the following methods : **swfmovie->output()**, **swfmovie->save()**, **swfmovie->add()**, **swfmovie->remove()**, **swfmovie->nextframe()**, **swfmovie->setbackground()**, **swfmovie->setrate()**, **swfmovie->setdimension()**, **swfmovie->setframes()** and **swfmovie->streammp3()**.

See examples in : **swfdisplayitem->rotateto()**, **swfshape->setline()**, **swfshape->addfill()**... Any example will use this object.

SWFMovie->add (unknown)

Adds any type of data to a movie.

void **swfmovie->add** (resource instance) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->add() adds *instance* to the current movie. *instance* is any type of data : Shapes, text, fonts, etc. must all be add'ed to the movie to make this work.

For displayable types (shape, text, button, sprite), this returns an **SWFDisplayItem()**, a handle to the object in a display list. Thus, you can add the same shape to a movie multiple times and get separate handles back for each separate instance.

See also all other objects (adding this later), and **swfmovie->remove()**

See examples in : **swfdisplayitem->rotateto()** and **swfshape->addfill()**.

SWFMovie->nextframe (unknown)

Moves to the next frame of the animation.

```
void swfmovie->nextframe ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->setframes() moves to the next frame of the animation.

SWFMovie->output (unknown)

Dumps your lovingly prepared movie out.

```
void swfmovie->output ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->output() dumps your lovingly prepared movie out. In PHP, preceding this with the command

```
<?php
header('Content-type: application/x-shockwave-flash');
?>
```

convinces the browser to display this as a flash movie.

See also **swfmovie->save()**.

See examples in : **swfmovie->streammp3()**, **swfdisplayitem->rotateto()**, **swfaction()**... Any example will use this method.

SWFMovie->remove (unknown)

Removes the object instance from the display list.

void **swfmovie->remove** (resource instance) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->remove() removes the object instance *instance* from the display list.

See also **swfmovie->add()**.

SWFMovie->save (unknown)

Saves your movie in a file.

void **swfmovie->save** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->save() saves your movie to the file named *filename*.

See also **output()**.

SWFMovie->setbackground (unknown)

Sets the background color.

void **swfmovie->setbackground** (int red, int green, int blue) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->setbackground() sets the background color. Why is there no rgba version? Think about it. (Actually, that's not such a dumb question after all- you might want to let the html background show through. There's a way to do that, but it only works on IE4. Search the <http://www.macromedia.com/> site for details.)

SWFMovie->setdimension (unknown)

Sets the movie's width and height.

```
void swfmovie->setdimension ( int width, int height) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->setdimension() sets the movie's width to *width* and height to *height*.

SWFMovie->setframes (unknown)

Sets the total number of frames in the animation.

```
void swfmovie->setframes ( string numberofframes) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->setframes() sets the total number of frames in the animation to *numberofframes*.

SWFMovie->setrate (unknown)

Sets the animation's frame rate.

```
void swfmovie->setrate ( int rate) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->setrate() sets the frame rate to *rate*, in frame per seconds. Animation will slow down if the player can't render frames fast enough- unless there's a streaming sound, in which case display frames are sacrificed to keep sound from skipping.

SWFMovie->streammp3 (unknown)

Streams a MP3 file.

```
void swfmovie->streammp3 ( string mp3FileName) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfmovie->streammp3() streams the mp3 file *mp3FileName*. Not very robust in dealing with oddities (can skip over an initial ID3 tag, but that's about it). Like **SWFShape->addJpegFill()**, this isn't a stable function- we'll probably need to make a separate SWFSound object to contain sound types.

Note that the movie isn't smart enough to put enough frames in to contain the entire mp3 stream- you'll have to add (length of song * frames per second) frames to get the entire stream in.

Yes, now you can use ming to put that rock and roll devil worship music into your SWF files. Just don't tell the RIAA.

Example 1. swfmovie->streammp3() example

```
<?php
    $m = new SWFMovie();
    $m->setRate(12.0);
    $m->streamMp3("distortobass.mp3");
    // use your own MP3

    // 11.85 seconds at 12.0 fps = 142 frames
    $m->setFrames(142);

    header('Content-type: application/x-shockwave-flash');
    $m->output();
?>
```

SWFShape (PHP 4 >= 4.0.5)

Creates a new shape object.

```
new swfshape ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape() creates a new shape object.

SWFShape has the following methods : **swfshape->setline()**, **swfshape->addfill()**, **swfshape->setleftfill()**, **swfshape->setrightfill()**, **swfshape->movepeno()**, **swfshape->movepen()**, **swfshape->drawlineto()**, **swfshape->drawline()**, **swfshape->drawcurveto()** and **swfshape->drawcurve()**.

This simple example will draw a big red elliptic quadrant.

Example 1. swfshape() example

```
<?php
    $s = new SWFShape();
    $s->setLine(40, 0x7f, 0, 0);
    $s->setRightFill($s->addFill(0xff, 0, 0));
    $s->movePenTo(200, 200);
    $s->drawLineTo(6200, 200);
    $s->drawLineTo(6200, 4600);
    $s->drawCurveTo(200, 4600, 200, 200);

    $m = new SWFMovie();
    $m->setDimension(6400, 4800);
    $m->setRate(12.0);
    $m->add($s);
    $m->nextFrame();

    header('Content-type: application/x-shockwave-flash');
    $m->output();
?>
```

SWFShape->addFill (unknown)

Adds a solid fill to the shape.

```
void swfshape->addfill ( int red, int green, int blue [, int a] )\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

```
void swfshape->addfill ( SWFbitmap bitmap [, int flags]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

```
void swfshape->addfill ( SWFGradient gradient [, int flags]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->addfill() adds a solid fill to the shape's list of fill styles. **swfshape->addfill()** accepts three different types of arguments.

red, *green*, *blue* is a color (RGB mode). Last parameter *a* is optional.

The *bitmap* argument is an `swfbitmap()` object. The *flags* argument can be one of the following values : `SWFFILL_CLIPPED_BITMAP` or `SWFFILL_TILED_BITMAP`. Default is `SWFFILL_TILED_BITMAP`. I think.

The *gradient* argument is an `swfgradient()` object. The *flags* argument can be one of the following values : `SWFFILL_RADIAL_GRADIENT` or `SWFFILL_LINEAR_GRADIENT`. Default is `SWFFILL_LINEAR_GRADIENT`. I'm sure about this one. Really.

swfshape->addfill() returns an `swffill()` object for use with the **swfshape->setleftfill()** and **swfshape->setrightfill()** functions described below.

See also **swfshape->setleftfill()** and **swfshape->setrightfill()**.

This simple example will draw a frame on a bitmap. Ah, here's another buglet in the flash player- it doesn't seem to care about the second shape's bitmap's transformation in a morph. According to spec, the bitmap should stretch along with the shape in this example..

Example 1. swfshape->addfill() example

```
<?php
    $p = new SWFMorph();

    $b = new SWFBitmap("alphafill.jpg");
```

```

// use your own bitmap
$width = $b->getWidth();
$height = $b->getHeight();

$s = $p->getShapel();
$f = $s->addFill($b, SWFFILL_TILED_BITMAP);
$f->moveTo(-$width/2, -$height/4);
$f->scaleTo(1.0, 0.5);
$s->setLeftFill($f);
$s->movePenTo(-$width/2, -$height/4);
$s->drawLine($width, 0);
$s->drawLine(0, $height/2);
$s->drawLine(-$width, 0);
$s->drawLine(0, -$height/2);

$s = $p->getShape2();
$f = $s->addFill($b, SWFFILL_TILED_BITMAP);

// these two have no effect!
$f->moveTo(-$width/4, -$height/2);
$f->scaleTo(0.5, 1.0);

$s->setLeftFill($f);
$s->movePenTo(-$width/4, -$height/2);
$s->drawLine($width/2, 0);
$s->drawLine(0, $height);
$s->drawLine(-$width/2, 0);
$s->drawLine(0, -$height);

$m = new SWFMovie();
$m->setDimension($width, $height);
$i = $m->add($p);
$i->moveTo($width/2, $height/2);

for($n=0; $n<1.001; $n+=0.03)
{
    $i->setRatio($n);
    $m->nextFrame();
}

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFShape->drawCurve (unknown)

Draws a curve (relative).

```
void swfshape->drawcurve ( int controldx, int controldy, int anchordx, int anchordy) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->drawcurve() draws a quadratic curve (using the current line style, set by **swfshape->setline()**) from the current pen position to the relative position (*anchorx, anchory*) using relative control point (*controlx, controly*). That is, head towards the control point, then smoothly turn to the anchor point.

See also **swfshape->drawlineto()**, **swfshape->drawline()**, **swfshape->movepen()** and **swfshape->movepen()**.

SWFShape->drawCurveTo (unknown)

Draws a curve.

```
void swfshape->drawcurveto ( int controlx, int controly, int anchorx, int anchory) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->drawcurveto() draws a quadratic curve (using the current line style, set by **swfshape->setline()**) from the current pen position to (*anchorx, anchory*) using (*controlx, controly*) as a control point. That is, head towards the control point, then smoothly turn to the anchor point.

See also **swfshape->drawlineto()**, **swfshape->drawline()**, **swfshape->movepen()** and **swfshape->movepen()**.

SWFShape->drawLine (unknown)

Draws a line (relative).

```
void swfshape->drawline ( int dx, int dy) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->drawline() draws a line (using the current line style set by **swfshape->setline()**) from the current pen position to displacement (dx,dy).

See also **swfshape->movepen()**, **swfshape->drawcurveto()**, **swfshape->movepen()** and **swfshape->drawlineto()**.

SWFShape->drawLineTo (unknown)

Draws a line.

```
void swfshape->drawlineto ( int x, int y) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->setrightfill() draws a line (using the current line style, set by **swfshape->setline()**) from the current pen position to point (x,y) in the shape's coordinate space.

See also **swfshape->movepen()**, **swfshape->drawcurveto()**, **swfshape->movepen()** and **swfshape->drawline()**.

SWFShape->movePen (unknown)

Moves the shape's pen (relative).

```
void swfshape->movepen ( int dx, int dy) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->setrightfill() move the shape's pen from coordinates (current x , current y) to (current $x + dx$, current $y + dy$) in the shape's coordinate space.

See also `swfshape->movepento()`, `swfshape->drawcurveto()`, `swfshape->drawlineto()` and `swfshape->drawline()`.

SWFShape->movePenTo (unknown)

Moves the shape's pen.

```
void swfshape->movepento ( int x, int y) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swfshape->setrightfill()` move the shape's pen to (x,y) in the shape's coordinate space.

See also `swfshape->movepen()`, `swfshape->drawcurveto()`, `swfshape->drawlineto()` and `swfshape->drawline()`.

SWFShape->setLeftFill (unknown)

Sets left rasterizing color.

```
void swfshape->setleftfill ( swfgradient fill) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

```
void swfshape->setleftfill ( int red, int green, int blue [, int a]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

What this nonsense is about is, every edge segment borders at most two fills. When rasterizing the object, it's pretty handy to know what those fills are ahead of time, so the swf format requires these to be specified.

swfshape->setleftfill() sets the fill on the left side of the edge- that is, on the interior if you're defining the outline of the shape in a counter-clockwise fashion. The fill object is an SWFFill object returned from one of the addFill functions above.

This seems to be reversed when you're defining a shape in a morph, though. If your browser crashes, just try setting the fill on the other side.

Shortcut for `swfshape->setleftfill($s->addfill($r, $g, $b [, $a]))`;

See also **swfshape->setrightfill()**.

SWFShape->setLine (unknown)

Sets the shape's line style.

```
void swfshape->setline ( int width [, int red [, int green [, int blue [, int a]]]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfshape->setline() sets the shape's line style. *width* is the line's width. If *width* is 0, the line's style is removed (then, all other arguments are ignored). If *width* > 0, then line's color is set to *red*, *green*, *blue*. Last parameter *a* is optional.

swfshape->setline() accepts 1, 4 or 5 arguments (not 3 or 2).

You must declare all line styles before you use them (see example).

This simple example will draw a big "!#%*@", in funny colors and gracious style.

Example 1. swfshape->setline() example

```
<?php
    $s = new SWFShape();
    $f1 = $s->addFill(0xff, 0, 0);
    $f2 = $s->addFill(0xff, 0x7f, 0);
    $f3 = $s->addFill(0xff, 0xff, 0);
    $f4 = $s->addFill(0, 0xff, 0);
    $f5 = $s->addFill(0, 0, 0xff);

    // bug: have to declare all line styles before you use them
    $s->setLine(40, 0x7f, 0, 0);
    $s->setLine(40, 0x7f, 0x3f, 0);
    $s->setLine(40, 0x7f, 0x7f, 0);
    $s->setLine(40, 0, 0x7f, 0);
    $s->setLine(40, 0, 0, 0x7f);
```



```

$f = new SWFFont('Techno.fdb');

$s->setRightFill($f1);
$s->setLine(40, 0x7f, 0, 0);
$s->drawGlyph($f, '!');
$s->movePen($f->getWidth('!'), 0);

$s->setRightFill($f2);
$s->setLine(40, 0x7f, 0x3f, 0);
$s->drawGlyph($f, '#');
$s->movePen($f->getWidth('#'), 0);

$s->setRightFill($f3);
$s->setLine(40, 0x7f, 0x7f, 0);
$s->drawGlyph($f, '%');
$s->movePen($f->getWidth('%'), 0);

$s->setRightFill($f4);
$s->setLine(40, 0, 0x7f, 0);
$s->drawGlyph($f, '*');
$s->movePen($f->getWidth('*'), 0);

$s->setRightFill($f5);
$s->setLine(40, 0, 0, 0x7f);
$s->drawGlyph($f, '@');

$m = new SWFMovie();
$m->setDimension(3000,2000);
$m->setRate(12.0);
$i = $m->add($s);
$i->moveTo(1500-$f->getWidth("!#%*@")/2, 1000+$f->getAscent()/2);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFShape->setRightFill (unknown)

Sets right rasterizing color.

```
void swfshape->setrightfill ( swfgradient fill) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

void **swfshape->setrightfill** (int red, int green, int blue [, int a]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

See also **swfshape->setleftfill**().

Shortcut for `swfshape->setrightfill($s->addfill($r, $g, $b [, $a]));`

SWFSprite (PHP 4 >= 4.0.5)

Creates a movie clip (a sprite)

new **swfsprite** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfsprite() are also known as a "movie clip", this allows one to create objects which are animated in their own timelines. Hence, the sprite has most of the same methods as the movie.

swfsprite() has the following methods : **swfsprite->add()**, **swfsprite->remove()**, **swfsprite->nextframe()** and **swfsprite->setframes()**.

This simple example will spin gracefully a big red square.

Example 1. swfsprite() example

```
<?php
    $s = new SWFShape();
    $s->setRightFill($s->addFill(0xff, 0, 0));
    $s->movePenTo(-500,-500);
    $s->drawLineTo(500,-500);
    $s->drawLineTo(500,500);
    $s->drawLineTo(-500,500);
```

```

$s->drawLineTo(-500,-500);

$p = new SWFSprite();
$i = $p->add($s);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();
$i->rotate(15);
$p->nextFrame();

$m = new SWFMovie();
$i = $m->add($p);
$i->moveTo(1500,1000);
$i->setName("blah");

$m->setBackground(0xff, 0xff, 0xff);
$m->setDimension(3000,2000);

header('Content-type: application/x-shockwave-flash');
$m->output();
?>

```

SWFSprite->add (unknown)

Adds an object to a sprite

void **swfsprite->add** (resource object) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfsprite->add() adds a swfshape(), a swfbutton(), a swftext(), a swfaction() or a swfsprite() object.

For displayable types (swfshape(), swfbutton(), swftext(), swfaction() or swfsprite()), this returns a handle to the object in a display list.

SWFSprite->nextframe (unknown)

Moves to the next frame of the animation.

```
void swfsprite->nextframe ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfsprite->setframes() moves to the next frame of the animation.

SWFSprite->remove (unknown)

Removes an object to a sprite

```
void swfsprite->remove ( ressource object) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfsprite->remove() remove a swfshape(), a swfbutton(), a swftext(), a swfaction() or a swfsprite() object from the sprite.

SWFSprite->setframes (unknown)

Sets the total number of frames in the animation.

```
void swfsprite->setframes ( int numberofframes) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swfsprite->setframes() sets the total number of frames in the animation to *numberofframes*.

SWFText (PHP 4 >= 4.0.5)

Creates a new SWFText object.

new **swftext** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext() creates a new SWFText object, fresh for manipulating.

SWFText has the following methods : **swftext->setfont()**, **swftext->setheight()**, **swftext->setspacing()**, **swftext->setcolor()**, **swftext->moveto()**, **swftext->addstring()** and **swftext->getwidth()**.

This simple example will draw a big yellow "PHP generates Flash with Ming" text, on white background.

Example 1. swftext() example

```

<?php
    $f = new SWFFont("Techno.fdb");
    $t = new SWFText();
    $t->setFont($f);
    $t->moveTo(200, 2400);
    $t->setColor(0xff, 0xff, 0);
    $t->setHeight(1200);
    $t->addString("PHP generates Flash with Ming!!");

    $m = new SWFMovie();
    $m->setDimension(5400, 3600);

    $m->add($t);

    header('Content-type: application/x-shockwave-flash');
    $m->output();
?>

```

SWFText->addString (unknown)

Draws a string

void **swftext->addstring** (string string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->addstring() draws the string *string* at the current pen (cursor) location. Pen is at the baseline of the text; i.e., ascending text is in the -y direction.

SWFText->getWidth (unknown)

Computes string's width

void **swftext->addstring** (string string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->addstring() returns the rendered width of the *string* string at the text object's current font, scale, and spacing settings.

SWFText->moveTo (unknown)

Moves the pen

void **swftext->moveto** (int x, int y) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->moveto() moves the pen (or cursor, if that makes more sense) to (x,y) in text object's coordinate space. If either is zero, though, value in that dimension stays the same. Annoying, should be fixed.

SWFText->setColor (unknown)

Sets the current font color

```
void swftext->setcolor ( int red, int green, int blue [, int a] )\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->setspacing() changes the current text color. Default is black. I think. Color is represented using the RGB system.

SWFText->setFont (unknown)

Sets the current font

```
void swftext->setfont ( string font )\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->setfont() sets the current font to *font*.

SWFText->setHeight (unknown)

Sets the current font height

```
void swftext->setheight ( int height )\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->setheight() sets the current font height to *height*. Default is 240.

SWFText->setSpacing (unknown)

Sets the current font spacing

```
void swftext->setspacing ( float spacing) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftext->setspacing() sets the current font spacing to *spacingspacing*. Default is 1.0. 0 is all of the letters written at the same point. This doesn't really work that well because it inflates the advance across the letter, doesn't add the same amount of spacing between the letters. I should try and explain that better, proly. Or just fix the damn thing to do constant spacing. This was really just a way to figure out how letter advances work, anyway.. So nyah.

SWFTextField (PHP 4 >= 4.0.5)

Creates a text field object

```
new swftextfield ( [int flags]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield() creates a new text field object. Text Fields are less flexible than swftext() objects- they can't be rotated, scaled non-proportionally, or skewed, but they can be used as form entries, and they can use browser-defined fonts.

The optional flags change the text field's behavior. It has the following possible values :

- SWFTEXTFIELD_NOEDIT indicates that the field shouldn't be user-editable
- SWFTEXTFIELD_PASSWORD obscures the data entry
- SWFTEXTFIELD_DRAWBOX draws the outline of the textfield
- SWFTEXTFIELD_MULTILINE allows multiple lines
- SWFTEXTFIELD_WORDWRAP allows text to wrap
- SWFTEXTFIELD_NOSELECT makes the field non-selectable

Flags are combined with the bitwise OR operation. For example,


```
<?php
$t = newSWFTextField(SWFTEXTFIELD_PASSWORD | SWFTEXTFIELD_NOEDIT);
?>
```

creates a totally useless non-editable password field.

SWFTextField has the following methods : `swftextfield->setfont()`, `swftextfield->setbounds()`, `swftextfield->align()`, `swftextfield->setheight()`, `swftextfield->setleftmargin()`, `swftextfield->setrightmargin()`, `swftextfield->setmargins()`, `swftextfield->setindentation()`, `swftextfield->setlinespacing()`, `swftextfield->setcolor()`, `swftextfield->setname()` and `swftextfield->addstring()`.

SWFTextField->addstring (unknown)

Concatenates the given string to the text field

```
void swftextfield->addstring ( string string ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swftextfield->setname()` concatenates the string *string* to the text field.

SWFTextField->align (unknown)

Sets the text field alignment

```
void swftextfield->align ( int alignment ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`swftextfield->align()` sets the text field alignment to *alignment*. Valid values for *alignment* are : `SWFTEXTFIELD_ALIGN_LEFT`, `SWFTEXTFIELD_ALIGN_RIGHT`, `SWFTEXTFIELD_ALIGN_CENTER` and `SWFTEXTFIELD_ALIGN_JUSTIFY`.

SWFTextField->setbounds (unknown)

Sets the text field width and height

```
void swftextfield->setbounds ( int width, int height) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setbounds() sets the text field width to *width* and height to *height*. If you don't set the bounds yourself, Ming makes a poor guess at what the bounds are.

SWFTextField->setcolor (unknown)

Sets the color of the text field.

```
void swftextfield->setcolor ( int red, int green, int blue [, int a]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setcolor() sets the color of the text field. Default is fully opaque black. Color is represented using RGB system.

SWFTextField->setFont (unknown)

Sets the text field font

```
void swftextfield->setfont ( string font) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setfont() sets the text field font to the [browser-defined?] *font* font.

SWFTextField->setHeight (unknown)

Sets the font height of this text field font.

```
void swftextfield->setheight ( int height) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setheight() sets the font height of this text field font to the given height *height*. Default is 240.

SWFTextField->setindentation (unknown)

Sets the indentation of the first line.

```
void swftextfield->setindentation ( int width) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setindentation() sets the indentation of the first line in the text field, to *width*.

SWFTextField->setLeftMargin (unknown)

Sets the left margin width of the text field.

```
void swftextfield->setleftmargin ( int width) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setleftmargin() sets the left margin width of the text field to *width*. Default is 0.

SWFTextField->setLineSpacing (unknown)

Sets the line spacing of the text field.

```
void swftextfield->setlinespacing ( int height) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setlinespacing() sets the line spacing of the text field to the height of *height*. Default is 40.

SWFTextField->setMargins (unknown)

Sets the margins width of the text field.

```
void swftextfield->setmargins ( int left, int right) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setmargins() set both margins at once, for the man on the go.

SWFTextField->setname (unknown)

Sets the variable name

```
void swftextfield->setname ( string name) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setname() sets the variable name of this text field to *name*, for form posting and action scripting purposes.

SWFTextField->setrightMargin (unknown)

Sets the right margin width of the text field.

```
void swftextfield->setrightmargin ( int width) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

swftextfield->setrightmargin() sets the right margin width of the text field to *width*. Default is 0.

LIX. 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 >= 4.0.0)

Returns `TRUE` if client disconnected

int **connection_aborted** (void) \linebreak

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 >= 4.0.0)

Returns connection status bitfield

int **connection_status** (void) \linebreak

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, 4.0.0 - 4.0.4 only)

Return `TRUE` if script timed out

bool **connection_timeout** (void) \linebreak

Returns `TRUE` if script timed out.

Deprecated

This function is deprecated, and doesn't even exist anymore as of 4.0.5.

See the Connection Handling description in the Features chapter for a complete explanation.

constant (PHP 4 >= 4.0.4)

Returns the value of a constant

mixed **constant** (string name) \linebreak

constant() will return the value of the constant indicated by *name*.

constant() is useful if you need to retrieve the value of a constant, but do not know it's name. i.e. It is stored in a variable or returned by a function.

Example 1. constant() example

```
<?php
define ("MAXSIZE", 100);

echo MAXSIZE;
echo constant("MAXSIZE"); // same thing as the previous line

?>
```

See also `define()`, `defined()` and the section on Constants.

define (PHP 3, PHP 4 >= 4.0.0)

Defines a named constant.

```
bool define ( string name, mixed value [, bool case_insensitive]) \linebreak
```

Defines a named constant. See the section on constants for more details.

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 `TRUE` 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."
echo Constant; // outputs "Constant" and issues a notice.

define ("GREETING", "Hello you.", TRUE);
echo GREETING; // outputs "Hello you."
echo Greeting; // outputs "Hello you."

?>
```

define() returns `TRUE` on success and `FALSE` if an error occurs.

See also `defined()`, `constant()` and the section on Constants.

defined (PHP 3, PHP 4 >= 4.0.0)

Checks whether a given named constant exists

bool **defined** (string *name*) \linebreak

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()`, `constant()`, `get_defined_constants()` and the section on Constants.

die (unknown)

Alias of `exit()`

This function is an alias of `exit()`.

eval (unknown)

Evaluate a string as PHP code

mixed **eval** (string *code_str*) \linebreak

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 immediately. In PHP 4, `eval()` returns `FALSE` unless `return()` is called in the evaluated code, in which case the value passed to `return()` is returned. In PHP 3, `eval()` does not return a value.

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.
```

Tip: As with anything that outputs its result directly to the browser, you can use the output-control functions to capture the output of this function, and save it - for example - in a string.

exit (unknown)

Output a message and terminate the current script

```
void exit ( [string status] ) \linebreak void exit ( int status ) \linebreak
```

Note: This is not a real function, but a language construct.

The `exit()` function terminates execution of the script. It prints `status` just before exiting.

If `status` is an integer, that value will also be used as the exit status. Exit statuses should be in the range 1 to 254, the exit status 255 is reserved by PHP and shall not be used.

Note: The current CVS version does NOT print the `status` if it is an integer.

Note: The `die()` function is an alias for **`exit()`**.

Example 1. `exit()` example

```
<?php

$filename = '/path/to/data-file';
$file = fopen ($filename, 'r')
    or exit("unable to open file ($filename)");

?>
```

get_browser (PHP 3, PHP 4 >= 4.0.0)

Tells what the user's browser is capable of

object **get_browser** ([string user_agent]) \linebreak

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 >= 4.0.0)

Syntax highlighting of a file

mixed **highlight_file** (string filename [, bool return]) \linebreak

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.

If the second parameter *return* is set to `TRUE` then **highlight_file()** will return the highlighted code as a string instead of printing it out. If the second parameter is not set to `TRUE` then **highlight_file()** will return `TRUE` on success, `FALSE` on failure.

Note: The *return* parameter became available in PHP 4.2.0. Before this time it behaved like the default, which is `FALSE`

Note: Care should be taken when using the `show_source()` and **highlight_file()** functions to make sure that you do not inadvertently reveal sensitive information such as passwords or any other type of information that might create a potential security risk.

Example 1. Creating a source highlighting URL

To setup a URL that can code highlight 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-httpd-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 colored 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 >= 4.0.0)

Syntax highlighting of a string

mixed **highlight_string** (string *str* [, bool *return*]) \linebreak

The **highlight_string()** function outputs a syntax highlighted version of *str* using the colors defined in the built-in syntax highlighter for PHP.

If the second parameter *return* is set to `TRUE` then **highlight_string()** will return the highlighted code as a string instead of printing it out. If the second parameter is not set to `TRUE` then **highlight_string()** will return `TRUE` on success, `FALSE` on failure.

Note: The *return* parameter became available in PHP 4.2.0. Before this time it behaved like the default, which is `FALSE`

See also `highlight_file()`, and `show_source()`.

ignore_user_abort (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Set whether a client disconnect should abort script execution

int **ignore_user_abort** ([int setting]) \linebreak

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 >= 4.0.0)

Parse a binary IPTC <http://www.iptc.org/> block into single tags.

array **iptcparse** (string iptcblock) \linebreak

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 >= 4.0.0)

Leak memory

void **leak** (int bytes) \linebreak

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 >= 4.0.0)

Pack data into binary string.

string **pack** (string format [, mixed args]) \linebreak

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 integer values as signed values of a machine dependent size. If you give it an unsigned integer value too large to be stored that way it is converted to a float which often yields an undesired result.

show_source (PHP 4 >= 4.0.0)

Syntax highlighting of a file

bool **show_source** (string filename [, bool return]) \linebreak

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.

If the second parameter *return* is set to `TRUE` then **show_source()** will return the highlighted code as a string instead of printing it out. If the second parameter is not set to `TRUE` then **show_source()** will return `TRUE` on success, `FALSE` on failure.

Note: The *return* parameter became available in PHP 4.2.0. Before this time it behaved like the default, which is `FALSE`

This function is an alias for the function `highlight_file()`

Note: Care should be taken when using the **show_source()** and `highlight_file()` functions to make sure that you do not inadvertently reveal sensitive information such as passwords or any other type of information that might create a potential security risk.

See also `highlight_string()`, `highlight_file()`.

sleep (PHP 3, PHP 4 >= 4.0.0)

Delay execution

void **sleep** (int seconds) \linebreak

The sleep function delays program execution for the given number of *seconds*.

See also `usleep()`.

uniqid (PHP 3, PHP 4 >= 4.0.0)

Generate a unique id

string **uniqid** (string prefix [, bool lcg]) \linebreak

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 prefix
$better_token = md5(uniqid(rand(),1)); // 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 >= 4.0.0)

Unpack data from binary string

array **unpack** (string format, string data) \linebreak

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 >= 4.0.0)

Delay execution in microseconds

```
void usleep ( int micro_seconds) \linebreak
```

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.

LX. mnoGoSearch Functions

These functions allow you to access mnoGoSearch (former UdmSearch) free search engine. In order to have these functions available, you must compile php with mnogosearch support by using the `--with-mnogosearch` option. If you use this option without specifying the path to mnogosearch, php will look for mnogosearch under `/usr/local/mnogosearch` path by default. If you installed mnogosearch at other path you should specify it: `--with-mnogosearch=DIR`.

mnoGoSearch is a full-featured search engine software for intranet and internet servers, distributed under the GNU license. mnoGoSearch has number of unique features, which makes it appropriate for a wide range of application from search within your site to a specialized search system such as cooking recipes or newspaper search, ftp archive search, news articles search, etc. It offers full-text indexing and searching for HTML, PDF, and text documents. mnoGoSearch consists of two parts. The first is an indexing mechanism (indexer). The purpose of indexer is to walk through HTTP, FTP, NEWS servers or local files, recursively grabbing all the documents and storing meta-data about that documents in a SQL database in a smart and effective manner. After every document is referenced by its corresponding URL, meta-data collected by indexer is used later in a search process. The search is performed via Web interface. C CGI, PHP and Perl search front ends are included.

Note: php contains built-in mysql access library, which can be used to access mysql. It is known that mnoGoSearch is not compatible with this built-in library and can work only with generic mysql libraries. Thus, if you use mnoGoSearch with mysql, during php configuration you have to indicate directory of mysql installation, that was used during mnoGoSearch configuration, i.e. for example:

```
--with-mnogosearch --with-mysql=/usr.
```

You need at least 3.1.10 version of mnoGoSearch installed to use these functions.

More information about mnoGoSearch can be found at <http://www.mnogosearch.ru/>.

udm_add_search_limit (PHP 4 >= 4.0.5)

Add various search limits

int **udm_add_search_limit** (int agent, int var, string val) \linebreak

udm_add_search_limit() returns `TRUE` on success, `FALSE` on error. Adds search restrictions.

agent - a link to Agent, received after call to `udm_alloc_agent()`.

var - defines parameter, indicating limit.

val - defines value of the current parameter.

Possible *var* values:

- `UDM_LIMIT_URL` - defines document URL limitations to limit search through subsection of database. It supports SQL `%` and `_` LIKE wildcards, where `%` matches any number of characters, even zero characters, and `_` matches exactly one character. E.g. `http://my.domain.__/catalog` may stand for `http://my.domain.ru/catalog` and `http://my.domain.ua/catalog`.
- `UDM_LIMIT_TAG` - defines site TAG limitations. In `indexer-conf` you can assign specific TAGs to various sites and parts of a site. Tags in `mnoGoSearch 3.1.x` are lines, that may contain metasymbols `%` and `_`. Metasymbols allow searching among groups of tags. E.g. there are links with tags `ABCD` and `ABCE`, and search restriction is by `ABC_` - the search will be made among both of the tags.
- `UDM_LIMIT_LANG` - defines document language limitations.
- `UDM_LIMIT_CAT` - defines document category limitations. Categories are similar to tag feature, but nested. So you can have one category inside another and so on. You have to use two characters for each level. Use a hex number going from 0-F or a 36 base number going from 0-Z. Therefore a top-level category like 'Auto' would be 01. If it has a subcategory like 'Ford', then it would be 01 (the parent category) and then 'Ford' which we will give 01. Put those together and you get 0101. If 'Auto' had another subcategory named 'VW', then it's id would be 01 because it belongs to the 'Ford' category and then 02 because it's the next category. So it's id would be 0102. If VW had a sub category called 'Engine' then it's id would start at 01 again and it would get the 'VW' id 02 and 'Auto' id of 01, making it 010201. If you want to search for sites under that category then you pass it `cat=010201` in the url.
- `UDM_LIMIT_DATE` - defines limitation by date document was modified.

Format of parameter value: a string with first character `<` or `>`, then with no space - date in unixtime format, for example:

```
Udm_Add_Search_Limit($udm,UDM_LIMIT_DATE,"<908012006");
```

If `>` character is used, then search will be restricted to those documents having modification date greater than entered. If `<`, then smaller.

udm_alloc_agent (PHP 4 >= 4.0.5)

Allocate mnoGoSearch session

```
int udm_alloc_agent ( string dbaddr [, string dbmode]) \linebreak
```

udm_alloc_agent() returns mnogosearch agent identifier on success, FALSE on error. This function creates a session with database parameters.

dbaddr - URL-style database description. Options (type, host, database name, port, user and password) to connect to SQL database. Do not matter for built-in text files support. Format: DBAddr DBType://[DBUser[:DBPass]@]DBHost[:DBPort]/DBName/ Currently supported DBType values are: mysql, pgsql, msql, solid, mssql, oracle, ibase. Actually, it does not matter for native libraries support. But ODBC users should specify one of supported values. If your database type is not supported, you may use "unknown" instead.

dbmode - You may select SQL database mode of words storage. When "single" is specified, all words are stored in the same table. If "multi" is selected, words will be located in different tables depending of their lengths. "multi" mode is usually faster but requires more tables in database. If "crc" mode is selected, mnoGoSearch will store 32 bit integer word IDs calculated by CRC32 algorithm instead of words. This mode requires less disk space and it is faster comparing with "single" and "multi" modes. "crc-multi" uses the same storage structure with the "crc" mode, but also stores words in different tables depending on words lengths like "multi" mode. Format: DBMode single/multi/crc/crc-multi

Note: *dbaddr* and *dbmode* must match those used during indexing.

Note: In fact this function does not open connection to database and thus does not check entered login and password. Actual connection to database and login/password verification is done by `udm_find()`.

udm_api_version (PHP 4 >= 4.0.5)

Get mnoGoSearch API version.

```
int udm_api_version ( void) \linebreak
```

udm_api_version() returns mnoGoSearch API version number. E.g. if mnoGoSearch 3.1.10 API is used, this function will return 30110.

This function allows user to identify which API functions are available, e.g. `udm_get_doc_count()` function is only available in mnoGoSearch 3.1.11 or later.

Example:

```
if (udm_api_version() >= 30111) {
```

```
print "Total number of urls in database: ".udm_get_doc_count($udm)."<br>\n";
}
```

udm_cat_list (PHP 4 >= 4.0.6)

Get all the categories on the same level with the current one.

array **udm_cat_list** (int agent, string category) \linebreak

udm_cat_list() returns array listing all categories of the same level as current category in the categories tree.

The function can be useful for developing categories tree browser.

Returns array with the following format:

The array consists of pairs. Elements with even index numbers contain category paths, odd elements contain corresponding category names.

```
$array[0] will contain '020300'
$array[1] will contain 'Audi'
$array[2] will contain '020301'
$array[3] will contain 'BMW'
$array[4] will contain '020302'
$array[5] will contain 'Opel'
...
etc.
```

Following is an example of displaying links of the current level in format:

```
Audi
BMW
Opel
...
```

```
<?php
$cat_list_arr = udm_cat_list($udm_agent,$cat);
$cat_list = "";
for ($i=0; $i<count($cat_list_arr); $i+=2) {
    $path = $cat_list_arr[$i];
    $name = $cat_list_arr[$i+1];
    $cat_list .= "<a href=\"\$PHP_SELF?cat=$path\">$name</a><br>";
}
}
```

?>

udm_cat_path (PHP 4 >= 4.0.6)

Get the path to the current category.

array **udm_cat_path** (int agent, string category) \linebreak

udm_cat_path() returns array describing path in the categories tree from the tree root to the current category.

agent - agent link identifier.

category - current category - the one to get path to.

Returns array with the following format:

The array consists of pairs. Elements with even index numbers contain category paths, odd elements contain corresponding category names.

For example, the call `$array=udm_cat_path($agent, '02031D');` may return the following array:

```
$array[0] will contain ""
$array[1] will contain 'Root'
$array[2] will contain '02'
$array[3] will contain 'Sport'
$array[4] will contain '0203'
$array[5] will contain 'Auto'
$array[4] will contain '02031D'
$array[5] will contain 'Ferrari'
```

Example 1. Specifying path to the current category in the following format: '> Root > Sport > Auto > Ferrari'

```
<?php
    $cat_path_arr = udm_cat_path($udm_agent,$cat);
    $cat_path = "";
    for ($i=0; $i<count($cat_path_arr); $i+=2) {
        $path = $cat_path_arr[$i];
        $name = $cat_path_arr[$i+1];
        $cat_path .= " > <a href=\"\$PHP_SELF?cat=$path\">$name</a> ";
    }
?>
```


udm_check_charset (PHP 4 >= 4.2.0)

Check if the given charset is known to mnogosearch

```
int udm_check_charset ( int agent, string charset) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

udm_check_stored (PHP 4 >= 4.2.0)

Check connection to stored

```
int udm_check_stored ( int agent, int link, string doc_id) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

udm_clear_search_limits (PHP 4 >= 4.0.5)

Clear all mnoGoSearch search restrictions

```
int udm_clear_search_limits ( int agent) \linebreak
```

udm_clear_search_limits() resets defined search limitations and returns TRUE.

udm_close_stored (PHP 4 >= 4.2.0)

Close connection to stored

```
int udm_close_stored ( int agent, int link) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

udm_crc32 (PHP 4 >= 4.2.0)

Return CRC32 checksum of gived string

```
int udm_crc32 ( int agent, string str) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

udm_errno (PHP 4 >= 4.0.5)

Get mnoGoSearch error number

```
int udm_errno ( int agent) \linebreak
```

udm_errno() returns mnoGoSearch error number, zero if no error.

agent - link to agent identifier, received after call to `udm_alloc_agent()`.

Receiving numeric agent error code.

udm_error (PHP 4 >= 4.0.5)

Get mnoGoSearch error message

```
string udm_error ( int agent) \linebreak
```

udm_error() returns mnoGoSearch error message, empty string if no error.

agent - link to agent identifier, received after call to `udm_alloc_agent()`.

Receiving agent error message.

udm_find (PHP 4 >= 4.0.5)

Perform search

```
int udm_find ( int agent, string query) \linebreak
```

udm_find() returns result link identifier on success, `FALSE` on error.

The search itself. The first argument - session, the next one - query itself. To find something just type words you want to find and press SUBMIT button. For example, "mysql odbc". You should not use quotes " in query, they are written here only to divide a query from other text. mnoGoSearch will find all documents that contain word "mysql" and/or word "odbc". Best documents having bigger weights will be displayed first. If you use search mode ALL, search will return documents that contain both (or more) words you entered. In case you use mode ANY, the search will return list of documents that contain any of the words you entered. If you want more advanced results you may use query language. You should select "bool" match mode in the search from.

mnoGoSearch understands the following boolean operators:

& - logical AND. For example, "mysql & odbc". mnoGoSearch will find any URLs that contain both "mysql" and "odbc".

| - logical OR. For example "mysql|odbc". mnoGoSearch will find any URLs, that contain word "mysql" or word "odbc".

~ - logical NOT. For example "mysql & ~odbc". mnoGoSearch will find URLs that contain word "mysql" and do not contain word "odbc" at the same time. Note that ~ just excludes given word from results. Query "~odbc" will find nothing!

() - group command to compose more complex queries. For example "(mysql | msq) & ~postgres". Query language is simple and powerful at the same time. Just consider query as usual boolean expression.

udm_free_agent (PHP 4 >= 4.0.5)

Free mnoGoSearch session

```
int udm_free_agent ( int agent) \linebreak
```

udm_free_agent() returns `TRUE` on success, `FALSE` on error.

agent - link to agent identifier, received ` after call to `udm_alloc_agent()`.

Freeing up memory allocated for agent session.

udm_free_ispell_data (PHP 4 >= 4.0.5)

Free memory allocated for ispell data

```
int udm_free_ispell_data ( int agent) \linebreak
```

udm_free_ispell_data() always returns `TRUE`.

agent - agent link identifier, received after call to `udm_alloc_agent()`.

Note: This function is supported beginning from version 3.1.12 of mnoGoSearch and it does not do anything in previous versions.

udm_free_res (PHP 4 >= 4.0.5)

Free mnoGoSearch result

int **udm_free_res** (int res) \linebreak

udm_free_res() returns `TRUE` on success, `FALSE` on error.

res - a link to result identifier, received after call to `udm_find()`.

Freeing up memory allocated for results.

udm_get_doc_count (PHP 4 >= 4.0.5)

Get total number of documents in database.

int **udm_get_doc_count** (int agent) \linebreak

udm_get_doc_count() returns number of documents in database.

agent - link to agent identifier, received after call to `udm_alloc_agent()`.

Note: This function is supported only in mnoGoSearch 3.1.11 or later.

udm_get_res_field (PHP 4 >= 4.0.5)

Fetch mnoGoSearch result field

string **udm_get_res_field** (int res, int row, int field) \linebreak

udm_get_res_field() returns result field value on success, `FALSE` on error.

res - a link to result identifier, received after call to `udm_find()`.

row - the number of the link on the current page. May have values from 0 to

`UDM_PARAM_NUM_ROWS-1`.

field - field identifier, may have the following values:

- UDM_FIELD_URL - document URL field
- UDM_FIELD_CONTENT - document Content-type field (for example, text/html).
- UDM_FIELD_CATEGORY - document category field. Use `udm_cat_path()` to get full path to current category from the categories tree root. (This parameter is available only in PHP 4.0.6 or later).
- UDM_FIELD_TITLE - document title field.
- UDM_FIELD_KEYWORDS - document keywords field (from META KEYWORDS tag).
- UDM_FIELD_DESC - document description field (from META DESCRIPTION tag).
- UDM_FIELD_TEXT - document body text (the first couple of lines to give an idea of what the document is about).
- UDM_FIELD_SIZE - document size.
- UDM_FIELD_URLID - unique URL ID of the link.
- UDM_FIELD_RATING - page rating (as calculated by mnoGoSearch).
- UDM_FIELD_MODIFIED - last-modified field in unixtime format.
- UDM_FIELD_ORDER - the number of the current document in set of found documents.
- UDM_FIELD_CRC - document CRC.

udm_get_res_param (PHP 4 >= 4.0.5)

Get mnoGoSearch result parameters

string **udm_get_res_param** (int res, int param) \linebreak

udm_get_res_param() returns result parameter value on success, FALSE on error.

res - a link to result identifier, received after call to `udm_find()`.

param - parameter identifier, may have the following values:

- UDM_PARAM_NUM_ROWS - number of received found links on the current page. It is equal to UDM_PARAM_PAGE_SIZE for all search pages, on the last page - the rest of links.
- UDM_PARAM_FOUND - total number of results matching the query.
- UDM_PARAM_WORDINFO - information on the words found. E.g. search for "a good book" will return "a: stopword, good:5637, book: 120"
- UDM_PARAM_SEARCHTIME - search time in seconds.
- UDM_PARAM_FIRST_DOC - the number of the first document displayed on current page.
- UDM_PARAM_LAST_DOC - the number of the last document displayed on current page.

udm_load_ispell_data (PHP 4 >= 4.0.5)

Load ispell data

int **udm_load_ispell_data** (int agent, int var, string val1, string val2, int flag) \linebreak

udm_load_ispell_data() loads ispell data. Returns `TRUE` on success, `FALSE` on error.

agent - agent link identifier, received after call to `udm_alloc_agent()`.

var - parameter, indicating the source for ispell data. May have the following values:

After using this function to free memory allocated for ispell data, please use `udm_free_ispell_data()`, even if you use `UDM_ISPELL_TYPE_SERVER` mode.

The fastest mode is `UDM_ISPELL_TYPE_SERVER`. `UDM_ISPELL_TYPE_TEXT` is slower and `UDM_ISPELL_TYPE_DB` is the slowest. The above pattern is `TRUE` for mnoGoSearch 3.1.10 - 3.1.11. It is planned to speed up DB mode in future versions and it is going to be faster than TEXT mode.

- `UDM_ISPELL_TYPE_DB` - indicates that ispell data should be loaded from SQL. In this case, parameters *val1* and *val2* are ignored and should be left blank. *flag* should be equal to 1.

Note: *flag* indicates that after loading ispell data from defined source it could be sorted (it is necessary for correct functioning of ispell). In case of loading ispell data from files there may be several calls to **udm_load_ispell_data()**, and there is no sense to sort data after every call, but only after the last one. Since in db mode all the data is loaded by one call, this parameter should have the value 1. In this mode in case of error, e.g. if ispell tables are absent, the function will return `FALSE` and code and error message will be accessible through `udm_error()` and `udm_errno()`.

Example:

```
if (! udm_load_ispell_data($udm,UDM_ISPELL_TYPE_DB,"",1)) {
    printf("Error #d: '%s'\n", udm_errno($udm), udm_error($udm));
    exit;
}
```

- `UDM_ISPELL_TYPE_AFFIX` - indicates that ispell data should be loaded from file and initiates loading affixes file. In this case *val1* defines double letter language code for which affixes are loaded, and *val2* - file path. Please note, that if a relative path entered, the module looks for the file not in `UDM_CONF_DIR`, but in relation to current path, i.e. to the path where the script is executed. In case of error in this mode, e.g. if file is absent, the function will return `FALSE`, and an error message will be displayed. Error message text cannot be accessed through `udm_error()` and `udm_errno()`, since those functions can only return messages associated with SQL. Please, see *flag* parameter description in `UDM_ISPELL_TYPE_DB`.

Example:

```

if ((! udm_load_ispell_data($udm,UDM_ISPELL_TYPE_AFFIX,'en','/opt/ispell/en.aff',0))
    (! udm_load_ispell_data($udm,UDM_ISPELL_TYPE_AFFIX,'ru','/opt/ispell/ru.aff',0)) |
    (! udm_load_ispell_data($udm,UDM_ISPELL_TYPE_SPELL,'en','/opt/ispell/en.dict',0))
    (! udm_load_ispell_data($udm,UDM_ISPELL_TYPE_SPELL,'ru','/opt/ispell/ru.dict',1))
exit;
}

```

Note: *flag* is equal to 1 only in the last call.

- UDM_ISPELL_TYPE_SPELL - indicates that ispell data should be loaded from file and initiates loading of ispell dictionary file. In this case *val1* defines double letter language code for which affixes are loaded, and *val2* - file path. Please note, that if a relative path entered, the module looks for the file not in UDM_CONF_DIR, but in relation to current path, i.e. to the path where the script is executed. In case of error in this mode, e.g. if file is absent, the function will return FALSE, and an error message will be displayed. Error message text cannot be accessed through `udm_error()` and `udm_erro()`, since those functions can only return messages associated with SQL. Please, see *flag* parameter description in UDM_ISPELL_TYPE_DB.

```

if ((! Udm_Load_Ispell_Data($udm,UDM_ISPELL_TYPE_AFFIX,'en','/opt/ispell/en.aff',0))
    (! Udm_Load_Ispell_Data($udm,UDM_ISPELL_TYPE_AFFIX,'ru','/opt/ispell/ru.aff',0)) |
    (! Udm_Load_Ispell_Data($udm,UDM_ISPELL_TYPE_SPELL,'en','/opt/ispell/en.dict',0))
    (! Udm_Load_Ispell_Data($udm,UDM_ISPELL_TYPE_SPELL,'ru','/opt/ispell/ru.dict',1))
exit;
}

```

Note: *flag* is equal to 1 only in the last call.

- UDM_ISPELL_TYPE_SERVER - enables spell server support. *val1* parameter indicates address of the host running spell server. *val2* ' is not used yet, but in future releases it is going to indicate number of port used by spell server. *flag* parameter in this case is not needed since ispell data is stored on spellserver already sorted.

Spelld server reads spell-data from a separate configuration file (`/usr/local/mnogosearch/etc/spelld.conf` by default), sorts it and stores in memory. With clients server communicates in two ways: to indexer all the data is transferred (so that indexer starts faster), from search.cgi server receives word to normalize and then passes over to client (search.cgi) list of normalized word forms. This allows fastest, compared to db and text modes processing of search queries (by omitting loading and sorting all the spell data).

udm_load_ispell_data() function in UDM_ISPELL_TYPE_SERVER mode does not actually load ispell data, but only defines server address. In fact, server is automatically used by `udm_find()` function when performing search. In case of errors, e.g. if spellserver is not running or invalid host indicated, there are no messages returned and ispell conversion does not work.

Note: This function is available in mnoGoSearch 3.1.12 or later.

Example:

```
if (!udm_load_ispell_data($udm,UDM_ISPELL_TYPE_SERVER,"",1)) {
    printf("Error loading ispell data from server<br>\n");
    exit;
}
```

udm_open_stored (PHP 4 >= 4.2.0)

Open connection to stored

int **udm_open_stored** (int agent, string storedaddr) \linebreak

Warning

This function is currently not documented; only the argument list is available.

udm_set_agent_param (PHP 4 >= 4.0.5)

Set mnoGoSearch agent session parameters

int **udm_set_agent_param** (int agent, int var, string val) \linebreak

udm_set_agent_param() returns `TRUE` on success, `FALSE` on error. Defines mnoGoSearch session parameters.

The following parameters and their values are available:

- `UDM_PARAM_PAGE_NUM` - used to choose search results page number (results are returned by pages beginning from 0, with `UDM_PARAM_PAGE_SIZE` results per page).

- UDM_PARAM_PAGE_SIZE - number of search results displayed on one page.
- UDM_PARAM_SEARCH_MODE - search mode. The following values available:
UDM_MODE_ALL - search for all words; UDM_MODE_ANY - search for any word;
UDM_MODE_PHRASE - phrase search; UDM_MODE_BOOL - boolean search. See `udm_find()` for details on boolean search.
- UDM_PARAM_CACHE_MODE - turns on or off search result cache mode. When enabled, the search engine will store search results to disk. In case a similar search is performed later, the engine will take results from the cache for faster performance. Available values: UDM_CACHE_ENABLED, UDM_CACHE_DISABLED.
- UDM_PARAM_TRACK_MODE - turns on or off trackquery mode. Since version 3.1.2 mnoGoSearch has a query tracking support. Note that tracking is implemented in SQL version only and not available in built-in database. To use tracking, you have to create tables for tracking support. For MySQL, use `create/mysql/track.txt`. When doing a search, front-end uses those tables to store query words, a number of found documents and current UNIX timestamp in seconds. Available values: UDM_TRACK_ENABLED, UDM_TRACK_DISABLED.
- UDM_PARAM_PHRASE_MODE - defines whether index database using phrases ("phrase" parameter in `indexer.conf`). Possible values: UDM_PHRASE_ENABLED and UDM_PHRASE_DISABLED. Please note, that if phrase search is enabled (UDM_PHRASE_ENABLED), it is still possible to do search in any mode (ANY, ALL, BOOL or PHRASE). In 3.1.10 version of mnoGoSearch phrase search is supported only in sql and built-in database modes, while beginning with 3.1.11 phrases are supported in cachemode as well.

Examples of phrase search:

"Arizona desert" - This query returns all indexed documents that contain "Arizona desert" as a phrase. Notice that you need to put double quotes around the phrase

- UDM_PARAM_CHARSET - defines local charset. Available values: set of charsets supported by mnoGoSearch, e.g. koi8-r, cp1251, ...
- UDM_PARAM_STOPFILE - Defines name and path to stopwords file. (There is a small difference with mnoGoSearch - while in mnoGoSearch if relative path or no path entered, it looks for this file in relation to UDM_CONF_DIR, the module looks for the file in relation to current path, i.e. to the path where the php script is executed.)
- UDM_PARAM_STOPTABLE - Load stop words from the given SQL table. You may use several StopwordTable commands. This command has no effect when compiled without SQL database support.
- UDM_PARAM_WEIGHT_FACTOR - represents weight factors for specific document parts. Currently body, title, keywords, description, url are supported. To activate this feature please use degrees of 2 in *Weight commands of the `indexer.conf`. Let's imagine that we have these weights:

```
URLWeight 1
BodyWeight 2
TitleWeight 4
KeywordWeight 8
DescWeight 16
```

As far as indexer uses bit OR operation for word weights when some word presents several time in the same document, it is possible at search time to detect word appearance in different document parts. Word which appears only in the body will have 00000010 argegate weight (in binary notation). Word used in all document parts will have 00011111 aggregate weight.

This parameter's value is a string of hex digits ABCDE. Each digit is a factor for corresponding bit in word weight. For the given above weights configuration:

- E is a factor for weight 1 (URL Weight bit)
- D is a factor for weight 2 (BodyWeight bit)
- C is a factor for weight 4 (TitleWeight bit)
- B is a factor for weight 8 (KeywordWeight bit)
- A is a factor for weight 16 (DescWeight bit)

Examples:

UDM_PARAM_WEIGHT_FACTOR=00001 will search through URLs only.

UDM_PARAM_WEIGHT_FACTOR=00100 will search through Titles only.

UDM_PARAM_WEIGHT_FACTOR=11100 will search through Title,Keywords,Description but not through URL and Body.

UDM_PARAM_WEIGHT_FACTOR=F9421 will search through:

- Description with factor 15 (F hex)
- Keywords with factor 9
- Title with factor 4
- Body with factor 2
- URL with factor 1

If UDM_PARAM_WEIGHT_FACTOR variable is ommitted, original weight value is taken to sort results. For a given above weight configuration it means that document description has a most big weight 16.

- UDM_PARAM_WORD_MATCH - word match. You may use this parameter to choose word match type. This feature works only in "single" and "multi" modes using SQL based and built-in database. It does not work in cachemode and other modes since they use word CRC and do not support substring search. Available values:

UDM_MATCH_BEGIN - word beginning match;

UDM_MATCH_END - word ending match;

UDM_MATCH_WORD - whole word match;

UDM_MATCH_SUBSTR - word substring match.

- UDM_PARAM_MIN_WORD_LEN - defines minimal word length. Any word shorter this limit is considered to be a stopword. Please note that this parameter value is inclusive, i.e. if UDM_PARAM_MIN_WORD_LEN=3, a word 3 characters long will not be considered a stopword, while a word 2 characters long will be. Default value is 1.

- UDM_PARAM_ISPELL_PREFIXES - Possible values: UDM_PREFIXES_ENABLED and UDM_PREFIXES_DISABLED, that respectively enable or disable using prefixes. E.g. if a word "tested" is in search query, also words like "test", "testing", etc. Only suffixes are supported by default. Prefixes usually change word meanings, for example if somebody is searching for the word "tested" one hardly wants "untested" to be found. Prefixes support may also be found useful for site's spelling checking purposes. In order to enable ispell, you have to load ispell data with `udm_load_ispell_data()`.
- UDM_PARAM_CROSS_WORDS - enables or disables crosswords support. Possible values: UDM_CROSS_WORDS_ENABLED and UDM_CROSS_WORDS_DISABLED.

The corsswords feature allows to assign words between `` and `` also to a document this link leads to. It works in SQL database mode and is not supported in built-in database and Cachemode.

Note: Crosswords are supported only in mnoGoSearch 3.1.11 or later.

- UDM_PARAM_VARDIR - specifies a custom path to directory where indexer stores data when using built-in database and in cache mode. By default `/var` directory of mnoGoSearch installation is used. Can have only string values. The parameter is available in PHP 4.1.0 or later.
- UDM_PARAM_VARDIR - specifies a custom path to directory where indexer stores data when using built-in database and in cache mode. By default `/var` directory of mnoGoSearch installation is used. Can have only string values. The parameter is available in PHP 4.1.0 or later.

LXI. 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/>.

mysql (PHP 3, PHP 4 >= 4.0.0)

Send mSQL query

```
int mysql ( string database, string query, int link_identifier) \linebreak
```

Returns a positive mSQL query identifier to the query result, or `FALSE` on error.

mysql() 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 `mysql_connect()` was called with no arguments (see `mysql_connect()`).

mysql_affected_rows (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Returns number of affected rows

```
int mysql_affected_rows ( int query_identifier) \linebreak
```

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: `mysql_query()`.

mysql_close (PHP 3, PHP 4 >= 4.0.0)

Close mSQL connection

```
int mysql_close ( int link_identifier) \linebreak
```

Returns `TRUE` on success, `FALSE` on error.

mysql_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.

mysql_close() will not close persistent links generated by `mysql_pconnect()`.

See also: `mysql_connect()` and `mysql_pconnect()`.

mysql_connect (PHP 3, PHP 4 >= 4.0.0)

Open mSQL connection

```
int mysql_connect ( [string hostname [, string server [, string username [, string password]]]]) \linebreak
```

Returns a positive mSQL link identifier on success, or `FALSE` on error.

mysql_connect() establishes a connection to a mSQL server. The *server* parameter can also include a port number. eg. "hostname:port". It defaults to 'localhost'.

In case 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()`.

See also `mysql_pconnect()`, `mysql_close()`.

mysql_create_db (PHP 3, PHP 4 >= 4.0.0)

Create mSQL database

```
int mysql_create_db ( string database name [, int link_identifier] ) \linebreak
```

`mysql_create_db()` attempts to create a new database on the server associated with the specified link identifier.

See also: `mysql_drop_db()`.

mysql_createdb (PHP 3, PHP 4 >= 4.0.0)

Create mSQL database

```
int mysql_createdb ( string database name [, int link_identifier] ) \linebreak
```

Identical to `mysql_create_db()`.

mysql_data_seek (PHP 3, PHP 4 >= 4.0.0)

Move internal row pointer

```
int mysql_data_seek ( int query_identifier, int row_number ) \linebreak
```

Returns `TRUE` on success, `FALSE` on failure.

mysql_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 `mysql_fetch_row()` would return that row.

See also: `mysql_fetch_row()`.

mysql_dbname (PHP 3, PHP 4 >= 4.0.0)

Get current mSQL database name

string **mysql_dbname** (int query_identifier, int i) \linebreak

mysql_dbname() returns the database name stored in position *i* of the result pointer returned from the `mysql_listdbs()` function. The `mysql_numrows()` function can be used to determine how many database names are available.

mysql_drop_db (PHP 3, PHP 4 >= 4.0.0)

Drop (delete) mSQL database

int **mysql_drop_db** (string database_name, int link_identifier) \linebreak

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()`.

mysql_dropdb (PHP 3, PHP 4 >= 4.0.0)

Drop (delete) mSQL database

See `mysql_drop_db()`.

mysql_error (PHP 3, PHP 4 >= 4.0.0)

Returns error message of last mysql call

string **mysql_error** ([int link_identifier]) \linebreak

Errors coming back from the mSQL database backend no longer issue warnings. Instead, use these functions to retrieve the error string.

mysql_fetch_array (PHP 3, PHP 4 >= 4.0.0)

Fetch row as array

int **mysql_fetch_array** (int query_identifier [, int result_type]) \linebreak

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.

The second optional argument *result_type* in **mysql_fetch_array()** is a constant and can take the following values: `MYSQL_ASSOC`, `MYSQL_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 **mysql_fetch_array()** is NOT significantly slower than using `mysql_fetch_row()`, while it provides a significant added value.

For further details, also see `mysql_fetch_row()`.

mysql_fetch_field (PHP 3, PHP 4 >= 4.0.0)

Get field information

object **mysql_fetch_field** (int query_identifier, int field_offset) \linebreak

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
- 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 `mysql_field_seek()`.

mysql_fetch_object (PHP 3, PHP 4 >= 4.0.0)

Fetch row as object

int **mysql_fetch_object** (int query_identifier [, int result_type]) \linebreak

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 second argument *result_type* in `mysql_fetch_array()` 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).

See also: `mysql_fetch_array()` and `mysql_fetch_row()`.

mysql_fetch_row (PHP 3, PHP 4 >= 4.0.0)

Get row as enumerated array

array **mysql_fetch_row** (int query_identifier) \linebreak

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 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 **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()`, and `mysql_result()`.

mysql_field_seek (PHP 3, PHP 4 >= 4.0.0)

Set field offset

int **mysql_field_seek** (int query_identifier, int field_offset) \linebreak

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()`.

mysql_fieldflags (PHP 3, PHP 4 >= 4.0.0)

Get field flags

string **mysql_fieldflags** (int query_identifier, int i) \linebreak

mysql_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 >= 4.0.0)

Get field length

```
int mysql_fieldlen ( int query_identifier, int i) \linebreak
```

mysql_fieldlen() returns the length of the specified field.

mysql_fieldname (PHP 3, PHP 4 >= 4.0.0)

Get field name

```
string mysql_fieldname ( int query_identifier, int field) \linebreak
```

mysql_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.

mysql_fieldtable (PHP 3, PHP 4 >= 4.0.0)

Get table name for field

```
int mysql_fieldtable ( int query_identifier, int field) \linebreak
```

Returns the name of the table *field* was fetched from.

mysql_fieldtype (PHP 3, PHP 4 >= 4.0.0)

Get field type

```
string mysql_fieldtype ( int query_identifier, int i) \linebreak
```

mysql_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".

mysql_free_result (PHP 3, PHP 4 >= 4.0.0)

Free result memory

```
int mysql_free_result ( int query_identifier) \linebreak
```

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_freeresult (PHP 3, PHP 4 >= 4.0.0)

Free result memory

See `mysql_free_result()`

mysql_list_dbs (PHP 3, PHP 4 >= 4.0.0)

List mSQL databases on server

```
int mysql_list_dbs ( void) \linebreak
```

mysql_list_dbs() will return a result pointer containing the databases available from the current msql daemon. Use the `mysql_dbname()` function to traverse this result pointer.

mysql_list_fields (PHP 3, PHP 4 >= 4.0.0)

List result fields

```
int mysql_list_fields ( string database, string tablename) \linebreak
```

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 a 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_list_tables (PHP 3, PHP 4 >= 4.0.0)

List tables in an mSQL database

```
int mysql_list_tables ( string database) \linebreak
```

mysql_list_tables() takes a database name and result pointer much like the `mysql()` function. The `mysql_tablename()` function should be used to extract the actual table names from the result pointer.

mysql_listdbs (PHP 3, PHP 4 >= 4.0.0)

List mSQL databases on server

See `mysql_list_dbs()`.

mysql_listfields (PHP 3, PHP 4 >= 4.0.0)

List result fields

See `mysql_list_fields()`.

mysql_listtables (PHP 3, PHP 4 >= 4.0.0)

List tables in an mSQL database

See `mysql_list_tables()`.

mysql_num_fields (PHP 3, PHP 4 >= 4.0.0)

Get number of fields in result

```
int mysql_num_fields ( int query_identifier) \linebreak
```

mysql_num_fields() returns the number of fields in a result set.

See also: `mysql()`, `mysql_query()`, `mysql_fetch_field()`, and `mysql_num_rows()`.

mysql_num_rows (PHP 3, PHP 4 >= 4.0.0)

Get number of rows in result

```
int mysql_num_rows ( int query_identifier) \linebreak
```

mysql_num_rows() returns the number of rows in a result set.

See also: `mysql()`, `mysql_query()`, and `mysql_fetch_row()`.

mysql_numfields (PHP 3, PHP 4 >= 4.0.0)

Get number of fields in result

```
int mysql_numfields ( int query_identifier) \linebreak
```

Identical to `mysql_num_fields()`.

mysql_numrows (PHP 3, PHP 4 >= 4.0.0)

Get number of rows in result

```
int mysql_numrows ( void) \linebreak
```

Identical to `mysql_num_rows()`.

mysql_pconnect (PHP 3, PHP 4 >= 4.0.0)

Open persistent mSQL connection

```
int mysql_pconnect ( [string server [, string username [, string password]]) \linebreak
```

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'.

mysql_query (PHP 3, PHP 4 >= 4.0.0)

Send mSQL query

```
int mysql_query ( string query, int link_identifier) \linebreak
```

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.

Example 1. mysql_query()

```
<?php
$link = mysql_connect("dbserver")
    or die("unable to connect to mysql server: ".mysql_error());
mysql_select_db("db", $link)
    or die("unable to select database 'db': ".mysql_error());

$result = mysql_query("SELECT * FROM table WHERE id=1", $link);
if (!$result) {
    die("query failed: ".mysql_error());
}

while ($row = mysql_fetch_array($result)) {
    echo $row["id"];
}
?>
```

See also: `mysql()`, `mysql_select_db()`, and `mysql_connect()`.

mysql_regcase (PHP 3, PHP 4 >= 4.0.0)

Make regular expression for case insensitive match

See `sql_regcase()`.

mysql_result (PHP 3, PHP 4 >= 4.0.0)

Get result data

```
int mysql_result ( int query_identifier, int i, mixed field) \linebreak
```

Returns the contents of the cell at the row and offset in the specified mSQL result set.

mysql_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 **mysql_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: **mysql_fetch_row()**, **mysql_fetch_array()**, and **mysql_fetch_object()**.

mysql_select_db (PHP 3, PHP 4 >= 4.0.0)

Select mSQL database

```
int mysql_select_db ( string database_name, int link_identifier ) \linebreak
```

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()**.

mysql_selectdb (PHP 3, PHP 4 >= 4.0.0)

Select mSQL database

See **mysql_select_db()**.

mysql_tablename (PHP 3, PHP 4 >= 4.0.0)

Get table name of field

```
string mysql_tablename ( int query_identifier, int field ) \linebreak
```

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_numrows()** function may be used to determine the number of tables in the result pointer.

Example 1. `mysql_tablename()` example

```
<?php
mysql_connect ("localhost");
$result = mysql_list_tables ("wisconsin");
$i = 0;
while ($i < mysql_numrows ($result)) {
    $tb_names[$i] = mysql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```


LXII. MySQL Functions

These functions allow you to access MySQL database servers. More information about MySQL can be found at <http://www.mysql.com/>.

Documentation for MySQL can be found at <http://www.mysql.com/documentation/>.

Requirements

In order to have these functions available, you must compile PHP with MySQL support.

Installation

By using the `--with-mysql` configuration option you enable PHP to access MySQL databases. If you use this option without specifying the path to MySQL, PHP will use the built-in MySQL client libraries. With PHP4 MySQL support is always enabled; if you don't specify the configure option, the bundled libraries are used. Users who run other applications that use MySQL (for example, running PHP 3 and PHP 4 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.

Runtime Configuration

The behaviour of the MySQL functions is affected by settings in the global configuration file `php.ini`.

Table 1. MySQL Configuration Options

Name	Default	Changeable
<code>mysql.allow_persistent</code>	"On"	PHP_INI_SYSTEM
<code>mysql.max_persistent</code>	"-1"	PHP_INI_SYSTEM
<code>mysql.max_links</code>	"-1"	PHP_INI_SYSTEM
<code>mysql.default_port</code>	NULL	PHP_INI_ALL
<code>mysql.default_socket</code>	NULL	PHP_INI_ALL
<code>mysql.default_host</code>	NULL	PHP_INI_ALL
<code>mysql.default_user</code>	NULL	PHP_INI_ALL
<code>mysql.default_password</code>	NULL	PHP_INI_ALL

For further details and definition of the PHP_INI_* constants see `ini_set()`.

Here is a short explanation of the configuration directives.

`mysql.allow_persistent` boolean

Whether to allow persistent connections to MySQL.

`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.

`mysql.default_port` string

The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the `MYSQL_TCP_PORT` environment variable, the `mysql-tcp` entry in `/etc/services` or the compile-time `MYSQL_PORT` constant, in that order. Win32 will only use the `MYSQL_PORT` constant.

`mysql.default_socket` string

The default socket name to use when connecting to a local database server if no other socket name is specified.

`mysql.default_host` string

The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in safe mode.

`mysql.default_user` string

The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in safe mode.

`mysql.default_password` string

The default password to use when connecting to the database server if no other password is specified. Doesn't apply in safe mode.

Resource types

There are two resource types used in the MySQL module. The first one is the link identifier for a

database connection, the second a resource which holds the result of a query.

Predefined constants

The function `mysql_fetch_array()` uses a constant for the different types of result arrays. The following constants are defined:

Table 2. MySQL fetch constants

constant	meaning
MYSQL_ASSOC	Columns are returned into the array having the fieldname as the array index.
MYSQL_BOTH	Columns are returned into the array having both a numerical index and the fieldname as the array index.
MYSQL_NUM	Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.
MYSQL_STORE_RESULT	Specifies that the MySQL result should be buffered.
MYSQL_USE_RESULT	Specifies that the MySQL result should not be buffered.

Examples

This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.

Example 1. MySQL extension overview example

```
<?php
/* Connecting, selecting database */
$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
    or die("Could not connect");
print "Connected successfully";
mysql_select_db("my_database") or die("Could not select database");

/* Performing SQL query */
$query = "SELECT * FROM my_table";
$result = mysql_query($query) or die("Query failed");

/* Printing results in HTML */
```

```
print "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    print "\t<tr>\n";
    foreach ($line as $col_value) {
        print "\t\t<td>$col_value</td>\n";
    }
    print "\t</tr>\n";
}
print "</table>\n";

/* Free resultset */
mysql_free_result($result);

/* Closing connection */
mysql_close($link);
?>
```

mysql_affected_rows (PHP 3, PHP 4 >= 4.0.0)

Get number of affected rows in previous MySQL operation

```
int mysql_affected_rows ( [resource link_identifier] ) \linebreak
```

mysql_affected_rows() returns the number of rows affected by the last INSERT, UPDATE or DELETE query associated with *link_identifier*. If the link identifier isn't specified, the last link opened by `mysql_connect()` 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.

Note: When using UPDATE, MySQL will not update columns where the new value is the same as the old value. This creates the possibility that **mysql_affected_rows()** may not actually equal the number of rows matched, only the number of rows that were literally affected by the query.

mysql_affected_rows() does not work with SELECT statements; only on statements which modify records. To retrieve the number of rows returned by a SELECT, use `mysql_num_rows()`.

If the last query failed, this function will return -1.

Example 1. Delete-Query

```
<?php
/* connect to database */
mysql_pconnect("localhost", "mysql_user", "mysql_password") or
    die ("Could not connect");

/* this should return the correct numbers of deleted records */
mysql_query("DELETE FROM mytable WHERE id < 10");
printf ("Records deleted: %d\n", mysql_affected_rows());

/* without a where clause in a delete statement, it should return 0 */
mysql_query("DELETE FROM mytable");
printf ("Records deleted: %d\n", mysql_affected_rows());
?>
```

The above example would produce the following output:

```
Records deleted: 10
Records deleted: 0
```

Example 2. Update-Query

```
<?php
    /* connect to database */
    mysql_pconnect("localhost", "mysql_user", "mysql_password") or
        die ("Could not connect");

    /* Update records */
    mysql_query("UPDATE mytable SET used=1 WHERE id < 10");
    printf ("Updated records: %d\n", mysql_affected_rows());
mysql_query("COMMIT");
?>
```

The above example would produce the following output:

```
Updated Records: 10
```

See also: `mysql_num_rows()`, `mysql_info()`.

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 [, resource link_identifier]]) \linebreak

mysql_change_user() changes the logged in user of the current active connection, or the connection given by the optional *link_identifier* parameter. If a database is specified, this will be the current database after the user has been changed. If the new user and password authorization fails, the current connected user stays active. Returns `TRUE` on success, `FALSE` on failure.

Note: This function was introduced in PHP 3.0.13 and requires MySQL 3.23.3 or higher. It is not available in PHP 4.

mysql_character_set_name (PHP 4 CVS only)

Returns the name of the character set

int **mysql_character_set_name** ([resource link_identifier]) \linebreak

mysql_character_set_name() returns the default character set name for the current connection.

Example 1. mysql_character_set_name() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$charset = mysql_character_set_name($link);
printf ("current character set is %s\n", $charset);
?>
```

The above example would produce the following output:

```
latin1
```

See also: [mysql_real_escape_string\(\)](#)

mysql_close (PHP 3, PHP 4 >= 4.0.0)

Close MySQL connection

bool **mysql_close** ([resource link_identifier]) \linebreak

Returns TRUE on success, FALSE on failure.

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. See also freeing resources.

Note: **mysql_close()** will not close persistent links created by [mysql_pconnect\(\)](#).

Example 1. MySQL close example

```
<?php
    $link = mysql_connect("localhost", "mysql_user", "mysql_password")
        or exit("Could not connect");
    print ("Connected successfully");
    mysql_close($link);
?>
```

See also: `mysql_connect()`, and `mysql_pconnect()`.

mysql_connect (PHP 3, PHP 4 >= 4.0.0)

Open a connection to a MySQL Server

resource **mysql_connect** ([string server [, string username [, string password [, bool new_link]]]]) \linebreak

Returns a MySQL link identifier on success, or `FALSE` on failure.

mysql_connect() establishes a connection to a MySQL server. The following defaults are assumed for missing optional parameters: *server* = 'localhost:3306', *username* = name of the user that owns the server process and *password* = empty password.

The *server* parameter 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 a @ 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 *new_link* parameter modifies this behavior and makes **mysql_connect()** always open a new link, even if **mysql_connect()** was called before with the same parameters.

Note: The *new_link* parameter became available in PHP 4.2.0

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
    $link = mysql_connect("localhost", "mysql_user", "mysql_password")
        or die("Could not connect");
    print ("Connected successfully");
    mysql_close($link);
?>
```

See also `mysql_pconnect()` and `mysql_close()`.

mysql_create_db (PHP 3, PHP 4 >= 4.0.0)

Create a MySQL database

bool **mysql_create_db** (string database name [, resource link_identifier]) \linebreak

mysql_create_db() attempts to create a new database on the server associated with the specified link identifier.

Returns TRUE on success, FALSE on failure.

Example 1. MySQL create database example

```
<?php
    $link = mysql_pconnect("localhost", "mysql_user", "mysql_password")
        or exit("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. This is deprecated, however.

Note: The function **mysql_create_db()** is deprecated. It is preferable to use `mysql_query()` to issue a SQL CREATE DATABASE Statement instead.

See also: `mysql_drop_db()`, `mysql_query()`.

mysql_data_seek (PHP 3, PHP 4 >= 4.0.0)

Move internal result pointer

```
bool mysql_data_seek ( resource result_identifier, int row_number) \linebreak
```

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 The *row_number* should be a value in the range from 0 to `mysql_num_rows - 1`.

Note: The function **mysql_data_seek()** can be used in conjunction only with `mysql_query()`, not with `mysql_unbuffered_query()`.

Example 1. MySQL data seek example

```
<?php
    $link = mysql_pconnect("localhost", "mysql_user", "mysql_password")
        or die("Could not connect");

    mysql_select_db("samp_db")
        or exit("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)) {
            echo "Cannot seek to row $i\n";
            continue;
        }

        if(!($row = mysql_fetch_object($result)))
            continue;

        echo "$row->last_name $row->first_name<br />\n";
    }

    mysql_free_result($result);
?>
```

See also: `mysql_query()`, `mysql_num_rows()`.

mysql_db_name (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Get result data

string **mysql_db_name** (resource result, int row [, mixed field]) \linebreak

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 >= 4.0.0)

Send a MySQL query

resource **mysql_db_query** (string database, string query [, resource link_identifier]) \linebreak

Returns a positive MySQL result resource 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()` and `mysql_query()`.

Note: This function has been deprecated since PHP 4.0.6. Do not use this function. Use `mysql_select_db()` and `mysql_query()` instead.

mysql_drop_db (PHP 3, PHP 4 >= 4.0.0)

Drop (delete) a MySQL database

```
bool mysql_drop_db ( string database_name [, resource link_identifier] ) \linebreak
```

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.

For downward compatibility **mysql_dropdb()** can also be used. This is deprecated, however.

Note: The function **mysql_drop_db()** is deprecated. It is preferable to use **mysql_query()** to issue a SQL DROP DATABASE statement instead.

See also: **mysql_create_db()** **mysql_query()**.

mysql_errno (PHP 3, PHP 4 >= 4.0.0)

Returns the numerical value of the error message from previous MySQL operation

```
int mysql_errno ( [resource link_identifier] ) \linebreak
```

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.

Example 1. mysql_errno Example

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password");

mysql_select_db("nonexistentdb");
echo mysql_errno() . ": " . mysql_error() . "\n";

mysql_select_db("kossu");
mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno() . ": " . mysql_error() . "\n";
?>
```

The above example would produce the following output:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See also: `mysql_error()`

mysql_error (PHP 3, PHP 4 >= 4.0.0)

Returns the text of the error message from previous MySQL operation

string **mysql_error** ([resource link_identifier]) \linebreak

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.

Example 1. mysql_errno Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password");

    mysql_select_db("nonexistentdb");
    echo mysql_errno() . ": " . mysql_error(). "\n";

    mysql_select_db("kossu");
    mysql_query("SELECT * FROM nonexistenttable");
    echo mysql_errno() . ": " . mysql_error() . "\n";
?>
```

The above example would produce the following output:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See also: `mysql_errno()`

`mysql_escape_string` (PHP 4 >= 4.0.3)

Escapes a string for use in a `mysql_query`.

string `mysql_escape_string` (string `unescaped_string`) \linebreak

This function will escape the *unescaped_string*, so that it is safe to place it in a `mysql_query`).

Note: `mysql_escape_string()` does not escape `%` and `_`.

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler and escapes the string according to the current character set.

`mysql_escape_string()` does not take a connection argument and does not respect the current charset setting.

Example 1. `mysql_real_escape_string()` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$item = "Zak's Laptop";
$escaped_item = mysql_real_escape_string($item);
printf ("Escaped string: %s\n", $escaped_item);
?>
```

The above example would produce the following output:

```
Escaped string: Zak\'s Laptop
```

See also `mysql_real_escape_string()`, `addslashes()`, and the `magic_quotes_gpc` directive.

`mysql_fetch_array` (PHP 3, PHP 4 >= 4.0.0)

Fetch a result row as an associative array, a numeric array, or both.

array `mysql_fetch_array` (resource `result` [, int `result_type`]) \linebreak

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. For aliased columns, you cannot access the contents with the original column name (by using `'field'` in this example).

Example 1. Query with duplicate field names

```
select table1.field as foo table2.field as bar from table1, table2
```

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. `MYSQL_BOTH` is the default for this argument.

By using `MYSQL_BOTH`, you'll get an array with both associative and number indices. Using `MYSQL_ASSOC`, you only get associative indices (as `mysql_fetch_assoc()` works), using `MYSQL_NUM`, you only get number indices (as `mysql_fetch_row()` works).

Example 2. mysql_fetch_array with MYSQL_NUM

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
    mysql_select_db("mydb");

    $result = mysql_query("SELECT id, name FROM mytable");

    while (($row = mysql_fetch_array($result, MYSQL_NUM))
        printf ("ID: %s Name: %s", $row[0], $row[1]);

    mysql_free_result($result);
?>
```

Example 3. mysql_fetch_array with MYSQL_ASSOC

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
```

```

mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while (($row = mysql_fetch_array($result, MYSQL_ASSOC))
    printf ("ID: %s Name: %s", $row["id"], $row["name"]);

mysql_free_result($result);
?>

```

Example 4. `mysql_fetch_array` with `MYSQL_BOTH`

```

<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("could not connect");
mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while (($row = mysql_fetch_array($result, MYSQL_BOTH))
    printf ("ID: %s Name: %s", $row[0], $row["name"]);

mysql_free_result($result);
?>

```

For further details, see also `mysql_fetch_row()` and `mysql_fetch_assoc()`.

`mysql_fetch_assoc` (PHP 4 >= 4.0.3)

Fetch a result row as an associative array

array **mysql_fetch_assoc** (resource result) \linebreak

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 either need to access the result with numeric indices by using `mysql_fetch_row()` or add alias names. See the example at the `mysql_fetch_array()` description about aliases.

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.

Example 1. mysql_fetch_assoc()

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password");
    mysql_select_db("mydb");
    $query = "select * from table";
    $result = mysql_query($query);
    while ($row = mysql_fetch_assoc($result)) {
        echo $row["user_id"];
        echo $row["fullname"];
    }
    mysql_free_result($result);
?>
```

For further details, see also `mysql_fetch_row()`, `mysql_fetch_array()` and `mysql_query()`.

mysql_fetch_field (PHP 3, PHP 4 >= 4.0.0)

Get column information from a result and return as an object

object **mysql_fetch_field** (resource result [, int field_offset]) \linebreak

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('localhost:3306', $user, $password)
    or die ("Could not connect");
mysql_select_db("database");
$result = mysql_query("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 >= 4.0.0)

Get the length of each output in a result

array **mysql_fetch_lengths** (resource result) \linebreak

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 >= 4.0.0)

Fetch a result row as an object

object **mysql_fetch_object** (resource result) \linebreak

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).

```
<?php
/* this is valid */
echo $row->field;
/* this is invalid */
echo $row->0;

?>
```

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("hostname", "user", "password");
mysql_select_db($db);
$result = mysql_query("select * from table");
while ($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 >= 4.0.0)

Get a result row as an enumerated array

array **mysql_fetch_row** (resource result) \linebreak

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 >= 4.0.0)

Get the flags associated with the specified field in a result

string **mysql_field_flags** (resource result, int field_offset) \linebreak

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. This is deprecated, however.

mysql_field_len (PHP 3, PHP 4 >= 4.0.0)

Returns the length of the specified field

int **mysql_field_len** (resource result, int field_offset) \linebreak

mysql_field_len() returns the length of the specified field.

For downward compatibility **mysql_fieldlen()** can also be used. This is deprecated, however.

mysql_field_name (PHP 3, PHP 4 >= 4.0.0)

Get the name of the specified field in a result

string **mysql_field_name** (resource result, int field_index) \linebreak

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.
 */
$link = mysql_connect('localhost', "mysql_user", "mysql_password");
mysql_select_db($dbname, $link)
    or die("Could not set $dbname");
$res = mysql_query("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. This is deprecated, however.

mysql_field_seek (PHP 3, PHP 4 >= 4.0.0)

Set result pointer to a specified field offset

int mysql_field_seek (resource result, int field_offset) \linebreak

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 >= 4.0.0)

Get name of the table the specified field is in

```
string mysql_field_table ( resource result, int field_offset) \linebreak
```

Returns the name of the table that the specified field is in.

For downward compatibility **mysql_fieldtable()** can also be used. This is deprecated, however.

mysql_field_type (PHP 3, PHP 4 >= 4.0.0)

Get the type of the specified field in a result

```
string mysql_field_type ( resource result, int field_offset) \linebreak
```

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", "mysql_username", "mysql_password");
mysql_select_db("mysql");
$result = mysql_query("SELECT * FROM func");
$fields = mysql_num_fields($result);
$rows   = mysql_num_rows($result);
$table  = mysql_field_table($result, 0);
echo "Your '". $table. "' table has ". $fields. " fields and ". $rows. " record(s)\n";
echo "The table has the following fields:\n";
for ($i=0; $i < $fields; $i++) {
    $type = mysql_field_type($result, $i);
    $name = mysql_field_name($result, $i);
    $len  = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type. " ". $name. " ". $len. " ". $flags. "\n";
}
mysql_free_result($result);
mysql_close();
?>
```

The above example would produce the following output:

```
Your 'func' table has 4 fields and 1 record(s)
The table has the following fields:
string name 64 not_null primary_key binary
```

```
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum
```

For downward compatibility `mysql_fieldtype()` can also be used. This is deprecated, however.

`mysql_free_result` (PHP 3, PHP 4 >= 4.0.0)

Free result memory

```
bool mysql_free_result ( resource result) \linebreak
```

`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.

Returns `TRUE` on success, `FALSE` on failure.

For downward compatibility `mysql_freeresult()` can also be used. This is deprecated, however.

`mysql_get_client_info` (PHP 4 >= 4.0.5)

Get MySQL client info

```
string mysql_get_client_info ( void) \linebreak
```

`mysql_get_client_info()` returns a string that represents the client library version.

Example 1. `mysql_get_client_info` Example

```
<?php
    printf ("MySQL client info: %s\n", mysql_get_client_info());
?>
```

The above example would produce the following output:

```
MySQL client info: 3.23.39
```

See also: `mysql_get_host_info()`, `mysql_get_proto_info()` and `mysql_get_server_info()`.

`mysql_get_host_info` (PHP 4 >= 4.0.5)

Get MySQL host info

string `mysql_get_host_info` ([resource `link_identifier`]) \linebreak

`mysql_get_host_info()` returns a string describing the type of connection in use for the connection `link_identifier`, including the server host name. If `link_identifier` is omitted, the last opened connection will be used.

Example 1. `mysql_get_host_info` Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
    printf ("MySQL host info: %s\n", mysql_get_host_info());
?>
```

The above example would produce the following output:

```
MySQL host info: Localhost via UNIX socket
```

See also: `mysql_get_client_info()`, `mysql_get_proto_info()` and `mysql_get_server_info()`.

`mysql_get_proto_info` (PHP 4 >= 4.0.5)

Get MySQL protocol info

int `mysql_get_proto_info` ([resource `link_identifier`]) \linebreak

`mysql_get_proto_info()` returns the protocol version used by connection `link_identifier`. If `link_identifier` is omitted, the last opened connection will be used.

Example 1. `mysql_get_proto_info` Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
    printf ("MySQL protocol version: %s\n", mysql_get_proto_info());
```



```
?>
```

The above example would produce the following output:

```
MySQL protocol version : 10
```

See also: `mysql_get_client_info()`, `mysql_get_host_info()` and `mysql_get_server_info()`.

mysql_get_server_info (PHP 4 >= 4.0.5)

Get MySQL server info

string **mysql_get_server_info** ([resource link_identifier]) \linebreak

mysql_get_server_info() returns the server version used by connection *link_identifier*. If *link_identifier* is omitted, the last opened connection will be used.

Example 1. mysql_get_server_info Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
    printf ("MySQL server version: %s\n", mysql_get_server_info());
?>
```

The above example would produce the following output:

```
MySQL server version: 4.0.1-alpha
```

See also: `mysql_get_client_info()`, `mysql_get_host_info()` and `mysql_get_proto_info()`.

mysql_info (PHP 4 CVS only)

Get information about the most recent query

string **mysql_info** ([resource link_identifier]) \linebreak

mysql_info() returns detailed information about the last query using the given *link_identifier*. If *link_identifier* isn't specified, the last opened link is assumed.

mysql_info() returns a string for all statements listed below. For all other FALSE. The string format depends on the given statement.

Example 1. Relevant MySQL Statements

```
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
String format: Records: 60 Duplicates: 0 Warnings: 0
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
```

The numbers are only for illustrating purpose; their values will correspond to the query.

Note: **mysql_info()** returns a non-FALSE value for the INSERT ... VALUES statement only if multiple value lists are specified in the statement.

See also: `mysql_affected_rows()`

mysql_insert_id (PHP 3, PHP 4 >= 4.0.0)

Get the id generated from the previous INSERT operation

```
int mysql_insert_id ( [resource link_identifier] ) \linebreak
```

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` (named `int` in PHP). 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()` in an SQL query.

Example 1. mysql_insert_id Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("could not connect");
    mysql_select_db("mydb");

    mysql_query("INSERT INTO mytable (product) values ('kossu');
    printf ("Last inserted record has id %d\n", mysql_insert_id());
?>
```

See also: `mysql_query()`.

mysql_list_dbs (PHP 3, PHP 4 >= 4.0.0)

List databases available on a MySQL server

resource **mysql_list_dbs** ([resource link_identifier]) \linebreak

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, or any function for result tables.

Example 1. mysql_list_dbs() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$db_list = mysql_list_dbs($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. This is deprecated however.

See also `mysql_db_name()`.

mysql_list_fields (PHP 3, PHP 4 >= 4.0.0)

List MySQL result fields

resource **mysql_list_fields** (string database_name, string table_name [, resource link_identifier]) \linebreak

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()`.

Example 1. mysql_list_fields() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');

$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. This is deprecated however.

mysql_list_processes (PHP 4 CVS only)

List MySQL processes

resource **mysql_list_processes** ([resource link_identifier]) \linebreak

mysql_list_processes() returns a result pointer describing the current server threads.

Example 1. mysql_list_processes() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');

$result = mysql_list_processes($link);
while ($row = mysql_fetch_row($result)){
    printf("%s %s %s %s %s\n", $row["Id"], $row["Host"], $row["db"],
        $row["Command"], $row["Time"]);
}
mysql_free_result ($result);
?>
```

The above example would produce the following output:

```
1 localhost test Processlist 0
4 localhost mysql sleep 5
```

See also: `mysql_thread_id()`

mysql_list_tables (PHP 3, PHP 4 >= 4.0.0)

List tables in a MySQL database

resource **mysql_list_tables** (string database [, resource link_identifier]) \linebreak

mysql_list_tables() takes a database name and returns a result pointer much like the `mysql_query()` function. You can use the `mysql_tablename()` function to extract the actual table names from the result pointer, or any other result table function.

For downward compatibility **mysql_listtables()** can also be used. This is deprecated however.

Example 1. mysql_list_tables Example

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("could not connect");
mysql_select_db("mydb");

$result = mysql_list_tables();

while (($row = mysql_fetch_row($result))
    printf ("Table: %s\n", $row[0]);

mysql_free_result($result);
?>
```

See also: `mysql_list_dbs()`, `mysql_tablename()`.

mysql_num_fields (PHP 3, PHP 4 >= 4.0.0)

Get number of fields in result

int **mysql_num_fields** (resource result) \linebreak

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. This is deprecated however.

mysql_num_rows (PHP 3, PHP 4 >= 4.0.0)

Get number of rows in result

int **mysql_num_rows** (resource result) \linebreak

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 affected by a INSERT, UPDATE or DELETE query, use `mysql_affected_rows()`.

Example 1. `mysql_num_rows()` example

```
<?php

$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("database", $link);

$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);

echo "$num_rows Rows\n";

?>
```

Note: If you use `mysql_unbuffered_query()`, **mysql_num_rows()** will not return the correct value until all the rows in the result set have been retrieved.

See also: `mysql_affected_rows()`, `mysql_connect()`, `mysql_data_seek()`, `mysql_select_db()`, and `mysql_query()`.

For downward compatibility **mysql_numrows()** can also be used. This is deprecated however.

mysql_pconnect (PHP 3, PHP 4 >= 4.0.0)

Open a persistent connection to a MySQL server

resource **mysql_pconnect** ([string server [, string username [, string password]]]) \linebreak

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: *server* = 'localhost:3306', *username* = name of the user that owns the server process and *password* = empty password.

The *server* parameter 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 link is therefore called 'persistent'.

Note: Note, that these kind of links only work if you are using a module version of PHP. See the Persistent Database Connections section for more information.

Warning

Using persistent connections can require a bit of tuning of your Apache and MySQL configurations to ensure that you do not exceed the number of connections allowed by MySQL.

mysql_ping (PHP 4 CVS only)

Ping a server connection or reconnect if there is no connection

```
bool mysql_ping ( [resource link_identifier] ) \linebreak
```

mysql_ping() checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted. This function can be used by scripts that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.

mysql_ping() returns `TRUE` if the connection to the server is working, otherwise `FALSE`.

See also: `mysql_thread_id()` `mysql_list_processes()`

mysql_query (PHP 3, PHP 4 >= 4.0.0)

Send a MySQL query

```
resource mysql_query ( string query [, resource link_identifier [, int result_mode]]) \linebreak
```

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.

The optional `result_mode` parameter can be `MYSQL_USE_RESULT` and `MYSQL_STORE_RESULT`. It defaults to `MYSQL_STORE_RESULT`, so the result is buffered. See also `mysql_unbuffered_query()` for the counterpart of this behaviour.

Note: The query string should not end with a semicolon.

Only for `SELECT`, `SHOW`, `EXPLAIN` or `DESCRIBE` statements `mysql_query()` returns a resource identifier or `FALSE` if the query was not executed correctly. For other type of SQL statements, `mysql_query()` returns `TRUE` on success and `FALSE` on error. A non-`FALSE` return value 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.

Only for SELECT, SHOW, DESCRIBE or EXPLAIN statements, **mysql_query()** returns a new result identifier that you can pass to **mysql_fetch_array()** and other functions dealing with result tables. 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_num_rows()**, **mysql_affected_rows()**, **mysql_unbuffered_query()**, **mysql_free_result()**, **mysql_fetch_array()**, **mysql_fetch_row()**, **mysql_fetch_assoc()**, **mysql_result()**, **mysql_select_db()**, and **mysql_connect()**.

mysql_real_escape_string (PHP 4 CVS only)

Escapes special characters in a string for use in a SQL statement, taking into account the current charset of the connection.

string **mysql_real_escape_string** (string *unescaped_string* [, resource *link_identifier*]) \linebreak

This function will escape special characters in the *unescaped_string*, taking into account the current charset of the connection so that it is safe to place it in a **mysql_query()**.

Note: **mysql_real_escape_string()** does not escape % and _.

Example 1. mysql_real_escape_string() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$item = "Zak's and Derick's Laptop";
$escaped_item = mysql_real_escape_string($item);
printf ("Escaped string: %s\n", $escaped_item);
?>
```

The above example would produce the following output:

```
Escaped string: Zak\'s and Derick\'s Laptop
```

See also: **mysql_escape_string()** **mysql_character_set_name()**

mysql_result (PHP 3, PHP 4 >= 4.0.0)

Get result data

mixed **mysql_result** (resource result, int row [, mixed field]) \linebreak

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 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 **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 >= 4.0.0)

Select a MySQL database

bool **mysql_select_db** (string database_name [, resource link_identifier]) \linebreak

Returns TRUE on success, FALSE on failure.

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 without arguments, 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. This is deprecated however.

mysql_stat (PHP 4 CVS only)

Get current system status

string **mysql_stat** ([resource link_identifier]) \linebreak

mysql_stat() returns the current server status.

Note: **mysql_stat()** currently only returns status for uptime, threads, queries, open tables, flush tables and queries per second. For a complete list of other status variables you have to use the SHOW STATUS sql command.

Example 1. mysql_stat() example

```
<?php
$link = mysql_connect('localhost', "mysql_user", "mysql_password");
printf("%s\n", mysql_stat($link));
?>
```

The above example would produce the following output:

```
Uptime: 5380  Threads: 1  Questions: 1321299  Slow queries: 1  Opens: 26  Flush ta-
bles: 1  Open tables: 17  Queries per second avg: 245.595
```

mysql_tablename (PHP 3, PHP 4 >= 4.0.0)

Get table name of field

string **mysql_tablename** (resource result, int i)\linebreak

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", "mysql_user", "mysql_password");
$result = mysql_list_tables("mydb");

for ($i = 0; $i < mysql_num_rows($result); $i++) {
    printf ("Table: %s\n", mysql_tablename($result, $i));
}

mysql_free_result($result);
?>
```

See also: `mysql_list_tables()`.

mysql_thread_id (PHP 4 CVS only)

Return the current thread id

int **mysql_thread_id** ([resource link_identifier]) \linebreak

mysql_thread_id() returns the current thread id. If the connection is lost and you reconnect with `mysql_ping()`, the thread ID will change. This means you should not get the thread ID and store it for later. You should get it when you need it.

Example 1. mysql_list_processes() example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$thread_id = mysql_thread_id($link);
if ($thread_id){
    printf ("current thread id is %d\n", $thread_id);
}
?>
```

The above example would produce the following output:

```
current thread id is 73
```

See also: `mysql_ping()` `mysql_list_processes()`

mysql_unbuffered_query (PHP 4 >= 4.0.6)

Send an SQL query to MySQL, without fetching and buffering the result rows

resource **mysql_unbuffered_query** (string query [, resource link_identifier [, int result_mode]]) \linebreak

mysql_unbuffered_query() sends a SQL query *query* to MySQL, without fetching and buffering the result rows automatically, as `mysql_query()` does. On the one hand, this saves a considerable amount of memory with SQL queries that produce large result sets. On the other hand, you can start working on the result set immediately after the first row has been retrieved: you don't have to wait until the complete

SQL query has been performed. When using multiple DB-connects, you have to specify the optional parameter *link_identifier*.

The optional *result_mode* parameter can be `MYSQL_USE_RESULT` and `MYSQL_STORE_RESULT`. It defaults to `MYSQL_USE_RESULT`, so the result is not buffered. See also `mysql_query()` for the counterpart of this behaviour.

Note: The benefits of `mysql_unbuffered_query()` come at a cost: You cannot use `mysql_num_rows()` on a result set returned from `mysql_unbuffered_query()`. You also have to fetch all result rows from an unbuffered SQL query, before you can send a new SQL query to MySQL.

See also: `mysql_query()`.

LXIII. Mohawk Software session handler functions

msession is an interface to a high speed session daemon which can run either locally or remotely. It is designed to provide consistent session management for a PHP web farm.

The session server software can be found at <http://www.mohawksoft.com/phoenix/>.

msession_connect (PHP 4 >= 4.2.0)

Connect to msession server

bool **msession_connect** (string host, string port) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_count (PHP 4 >= 4.2.0)

Get session count

int **msession_count** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_create (PHP 4 >= 4.2.0)

Create a session

bool **msession_create** (string session) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_destroy (PHP 4 >= 4.2.0)

Destroy a session

bool **msession_destroy** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_disconnect (PHP 4 >= 4.2.0)

Close connection to msession server

void **msession_disconnect** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_find (PHP 4 >= 4.2.0)

Find value

array **msession_find** (string name, string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_get (PHP 4 >= 4.2.0)

Get value from session

string **msession_get** (string session, string name, string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_get_array (PHP 4 >= 4.2.0)

Get array of ... ?

array **msession_get_array** (string session) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_getdata (unknown)

Get data ... ?

string **msession_getdata** (string session) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_inc (PHP 4 >= 4.2.0)

Increment value in session

string **msession_inc** (string session, string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_list (PHP 4 >= 4.2.0)

List ... ?

array **msession_list** (void) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_listvar (PHP 4 >= 4.2.0)

List sessions with variable

array **msession_listvar** (string name) \linebreak

Returns an associative array of value, session for all sessions with a variable named *name*.

Used for searching sessions with common attributes.

msession_lock (PHP 4 >= 4.2.0)

Lock a session

int **msession_lock** (string name) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_plugin (PHP 4 >= 4.2.0)

Call an escape function within the msession personality plugin

string **msession_plugin** (string session, string val [, string param]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_randstr (PHP 4 >= 4.2.0)

Get random string

string **msession_randstr** (int param) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_set (PHP 4 >= 4.2.0)

Set value in session

bool **msession_set** (string session, string name, string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_set_array (PHP 4 >= 4.2.0)

Set array of ...

bool **msession_set_array** (string session, array tuples) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_setdata (unknown)

Set data ... ?

bool **msession_setdata** (string session, string value) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_timeout (PHP 4 >= 4.2.0)

Set/get session timeout

int **msession_timeout** (string session [, int param]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_uniq (PHP 4 >= 4.2.0)

Get uniq id

string **msession_uniq** (int param) \linebreak

Warning

This function is currently not documented; only the argument list is available.

msession_unlock (PHP 4 >= 4.2.0)

Unlock a session

int **msession_unlock** (string session, int key) \linebreak

Warning

This function is currently not documented; only the argument list is available.

LXIV. muscat functions

muscat_close (4.0.5 - 4.2.0 only)

Shuts down the muscat session and releases any memory back to php. [Not back to the system, note!]

int **muscat_close** (resource muscat_handle) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

muscat_get (4.0.5 - 4.2.0 only)

Gets a line back from the core muscat api. Returns a literal FALSE when there is no more to get (as opposed to ""). Use === FALSE or !== FALSE to check for this

string **muscat_get** (resource muscat_handle) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

muscat_give (4.0.5 - 4.2.0 only)

Sends string to the core muscat api

int **muscat_give** (resource muscat_handle, string string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

muscat_setup (4.0.5 - 4.2.0 only)

Creates a new muscat session and returns the handle. Size is the amount of memory in bytes to allocate for muscat muscat_dir is the muscat installation dir e.g. "/usr/local/empower", it defaults to the compile time muscat directory

resource **muscat_setup** (int size [, string muscat_dir]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

muscat_setup_net (4.0.5 - 4.2.0 only)

Creates a new muscat session and returns the handle. muscat_host is the hostname to connect to port is the port number to connect to - actually takes exactly the same args as fsockopen

resource **muscat_setup_net** (string muscat_host, int port) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

LXV. Network Functions

checkdnsrr (PHP 3, PHP 4 >= 4.0.0)

Check DNS records corresponding to a given Internet host name or IP address

int **checkdnsrr** (string host [, string type]) \linebreak

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.

Note: This function is not implemented on Windows platforms.

See also `getmxrr()`, `gethostbyaddr()`, `gethostbyname()`, `gethostbynameel()`, and the `named(8)` manual page.

closelog (PHP 3, PHP 4 >= 4.0.0)

Close connection to system logger

int **closelog** (void) \linebreak

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) \linebreak

Disables the internal PHP debugger. The debugger is still under development.

debugger_on (PHP 3)

Enable internal PHP debugger

int **debugger_on** (string address) \linebreak

Enables the internal PHP debugger, connecting it to *address*. The debugger is still under development.

define_syslog_variables (PHP 3, PHP 4 >= 4.0.0)

Initializes all syslog related constants

```
void define_syslog_variables ( void) \linebreak
```

Initializes all constants used in the syslog functions.

See also `openlog()`, `syslog()` and `closelog()`.

fsockopen (PHP 3, PHP 4 >= 4.0.0)

Open Internet or Unix domain socket connection

```
int fsockopen ( string hostname, int port [, int errno [, string errstr [, float timeout]]]) \linebreak
```

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 by prefixing *hostname* with 'udp://'. 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.

As of PHP 4.3.0, if you have compiled in OpenSSL support, you may prefix the *hostname* with either 'ssl://' or 'tls://' to use an SSL or TLS client connection over TCP/IP to connect to the remote host.

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 in the system-level `connect()` call. If the value returned in *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 will always 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

```
<?php
$fp = fsockopen ("www.example.com", 80, $errno, $errstr, 30);
if (!$fp) {
    echo "$errstr ($errno)<br>\n";
} else {
    fputs ($fp, "GET / HTTP/1.0\r\nHost: www.example.com\r\n\r\n");
    while (!feof($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()`, `feof()`, and the Curl extension.

gethostbyaddr (PHP 3, PHP 4 >= 4.0.0)

Get the Internet host name corresponding to a given IP address

string **gethostbyaddr** (string *ip_address*) \linebreak

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 >= 4.0.0)

Get the IP address corresponding to a given Internet host name

string **gethostbyname** (string hostname) \linebreak

Returns the IP address of the Internet host specified by *hostname*.

See also `gethostbyaddr()`.

gethostbynameall (PHP 3, PHP 4 >= 4.0.0)

Get a list of IP addresses corresponding to a given Internet host name

array **gethostbynameall** (string hostname) \linebreak

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 >= 4.0.0)

Get MX records corresponding to a given Internet host name

int **getmxrr** (string hostname, array mxhosts [, array weight]) \linebreak

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()`, `gethostbynameall()`, `gethostbyaddr()`, and the `named(8)` manual page.

getprotobyname (PHP 4 >= 4.0.0)

Get protocol number associated with protocol name

int **getprotobyname** (string name) \linebreak

getprotobyname() returns the protocol number associated with the protocol *name* as per `/etc/protocols`.

See also: `getprotobyname()`.

getprotobynumber (PHP 4 >= 4.0.0)

Get protocol name associated with protocol number

string **getprotobynumber** (int number) \linebreak

getprotobynumber() returns the protocol name associated with protocol *number* as per */etc/protocols*.

See also: `getprotobyname()`.

getservbyname (PHP 4 >= 4.0.0)

Get port number associated with an Internet service and protocol

int **getservbyname** (string service, string protocol) \linebreak

getservbyname() returns the Internet port which corresponds to *service* for the specified *protocol* as per */etc/services*. *protocol* is either "tcp" or "udp" (in lowercase).

See also: `getservbyport()`.

getservbyport (PHP 4 >= 4.0.0)

Get Internet service which corresponds to port and protocol

string **getservbyport** (int port, string protocol) \linebreak

getservbyport() returns the Internet service associated with *port* for the specified *protocol* as per */etc/services*. *protocol* is either "tcp" or "udp" (in lowercase).

See also: `getservbyname()`.

ip2long (PHP 4 >= 4.0.0)

Converts a string containing an (IPv4) Internet Protocol dotted address into a proper address.

int **ip2long** (string ip_address) \linebreak

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.example.com");
$out = "The following URLs are equivalent:<br>\n";
$out .= "http://www.example.com/, http://".$ip."/, and http://".sprintf("%u",ip2long($ip)).".";
echo $out;
?>
```

Note: Because PHP's integer type is signed, and many IP addresses will result in negative integers, you need to use the "%u" formatter of sprintf() or printf() to get the string representation of the unsigned IP address.

This second example shows how to print a converted address with the printf() function :

Example 2. Displaying an IP address

```
<?php
$ip = gethostbyname("www.example.com");
printf("%u\n", ip2long($ip));
echo $out;
?>
```

See also: long2ip()

long2ip (PHP 4 >= 4.0.0)

Converts an (IPv4) Internet network address into a string in Internet standard dotted format

string **long2ip** (int proper_address) \linebreak

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 >= 4.0.0)

Open connection to system logger

int **openlog** (string *ident*, int *option*, int *facility*) \linebreak

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 logging 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 opening 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
LOG_AUTH	security/authorization messages (use LOG_AUTHPRIV instead in systems where that constant is defined)
LOG_AUTHPRIV	security/authorization messages (private)
LOG_CRON	clock daemon (cron and at)
LOG_DAEMON	other system daemons
LOG_KERN	kernel messages
LOG_LOCAL0 ... LOG_LOCAL7	reserved for local use
LOG_LPR	line printer subsystem
LOG_MAIL	mail subsystem
LOG_NEWS	USENET news subsystem
LOG_SYSLOG	messages generated internally by syslogd
LOG_USER	generic user-level messages
LOG_UUCP	UUCP subsystem

See also `define_syslog_variables()`, `syslog()` and `closelog()`.

psockopen (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Open persistent Internet or Unix domain socket connection

`int psockopen (string hostname, int port [, int errno [, string errstr [, int timeout]]])` \linebreak

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 >= 4.0.0)

Returns information about an existing socket stream

`array socket_get_status (resource socketstream)` \linebreak

Returns information about an existing socket stream. This function only works on sockets created by `fsockopen()`, `psockopen()` or network sockets returned by `fopen()` when opening URLs. It does NOT work with sockets from the Socket extension. 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: Socket extension.

socket_set_blocking (PHP 4 >= 4.0.0)

Set blocking/non-blocking mode on a socket

`int socket_set_blocking (int socket descriptor, int mode)` \linebreak

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 >= 4.0.0)

Set timeout period on a socket

bool **socket_set_timeout** (int socket descriptor, int seconds, int microseconds) \linebreak

Sets the timeout value on *socket descriptor*, expressed in the sum of *seconds* and *microseconds*.

Example 1. socket_set_timeout() Example

```
<?php
$fp = fsockopen("www.example.com", 80);
if(!$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 >= 4.0.0)

Generate a system log message

int **syslog** (int priority, string message) \linebreak

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.

Constant	Description
----------	-------------

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()`.

LXVI. Ncurses terminal screen control functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

What is ncurses?

ncurses (new curses) is a free software emulation of curses in System V Rel 4.0 (and above). It uses terminfo format, supports pads, colors, multiple highlights, form characters and function key mapping.

Platforms

Ncurses is available for the following platforms:

- AIX
- BeOS
- Cygwin
- Digital Unix (aka OSF1)
- FreeBSD
- GNU/Linux
- HPUX
- IRIX
- OS/2
- SCO OpenServer
- Solaris
- SunOS

Requirements

You need the ncurses libraries and headerfiles. Download the latest version from the

<ftp://ftp.gnu.org/pub/gnu/ncurses/> or from an other GNU-Mirror.

Installation

To get these functions to work, you have to compile the CGI version of PHP with `--with-ncurses`.

Ncurses predefined constants

Error codes

On error ncurses functions return `NCURSES_ERR`.

Colors

Table 1. ncurses color constants

constant	meaning
<code>NCURSES_COLOR_BLACK</code>	no color (black)
<code>NCURSES_COLOR_WHITE</code>	white
<code>NCURSES_COLOR_RED</code>	red - supported when terminal is in color mode
<code>NCURSES_COLOR_GREEN</code>	green - supported when terminal is in color mod
<code>NCURSES_COLOR_YELLOW</code>	yellow - supported when terminal is in color mod
<code>NCURSES_COLOR_BLUE</code>	blue - supported when terminal is in color mod
<code>NCURSES_COLOR_CYAN</code>	cyan - supported when terminal is in color mod
<code>NCURSES_COLOR_MAGENTA</code>	magenta - supported when terminal is in color mod

Keys

Table 2. ncurses key constants

constant	meaning
<code>NCURSES_KEY_F0 - NCURSES_KEY_F64</code>	function keys F1 - F64
<code>NCURSES_KEY_DOWN</code>	down arrow
<code>NCURSES_KEY_UP</code>	up arrow
<code>NCURSES_KEY_LEFT</code>	left arrow
<code>NCURSES_KEY_RIGHT</code>	right arrow
<code>NCURSES_KEY_HOME</code>	home key (upward+left arrow)

constant	meaning
NCURSES_KEY_BACKSPACE	backspace
NCURSES_KEY_DL	delete line
NCURSES_KEY_IL	insert line
NCURSES_KEY_DC	delete character
NCURSES_KEY_IC	insert char or enter insert mode
NCURSES_KEY_EIC	exit insert char mode
NCURSES_KEY_CLEAR	clear screen
NCURSES_KEY_EOS	clear to end of screen
NCURSES_KEY_EOL	clear to end of line
NCURSES_KEY_SF	scroll one line forward
NCURSES_KEY_SR	scroll one line backward
NCURSES_KEY_NPAGE	next page
NCURSES_KEY_PPAGE	previous page
NCURSES_KEY_STAB	set tab
NCURSES_KEY_CTAB	clear tab
NCURSES_KEY_CATAB	clear all tabs
NCURSES_KEY_SRESET	soft (partial) reset
NCURSES_KEY_RESET	reset or hard reset
NCURSES_KEY_PRINT	print
NCURSES_KEY_LL	lower left
NCURSES_KEY_A1	upper left of keypad
NCURSES_KEY_A3	upper right of keypad
NCURSES_KEY_B2	center of keypad
NCURSES_KEY_C1	lower left of keypad
NCURSES_KEY_C3	lower right of keypad
NCURSES_KEY_BTAB	back tab
NCURSES_KEY_BEG	beginning
NCURSES_KEY_CANCEL	cancel
NCURSES_KEY_CLOSE	close
NCURSES_KEY_COMMAND	cmd (command)
NCURSES_KEY_COPY	copy
NCURSES_KEY_CREATE	create
NCURSES_KEY_END	end
NCURSES_KEY_EXIT	exit
NCURSES_KEY_FIND	find
NCURSES_KEY_HELP	help
NCURSES_KEY_MARK	mark
NCURSES_KEY_MESSAGE	message

constant	meaning
NCURSES_KEY_MOVE	move
NCURSES_KEY_NEXT	next
NCURSES_KEY_OPEN	open
NCURSES_KEY_OPTIONS	options
NCURSES_KEY_PREVIOUS	previous
NCURSES_KEY_REDO	redo
NCURSES_KEY_REFERENCE	ref (reference)
NCURSES_KEY_REFRESH	refresh
NCURSES_KEY_REPLACE	replace
NCURSES_KEY_RESTART	restart
NCURSES_KEY_RESUME	resume
NCURSES_KEY_SAVE	save
NCURSES_KEY_SBEG	shifet beg (beginning)
NCURSES_KEY_SCANCEL	shifted cancel
NCURSES_KEY_SCOMMAND	shifted command
NCURSES_KEY_SCOPY	shifted copy
NCURSES_KEY_SCREATE	shifted create
NCURSES_KEY_SDC	shifted delete char
NCURSES_KEY_SDL	shifted delete line
NCURSES_KEY_SELECT	select
NCURSES_KEY_SEND	shifted end
NCURSES_KEY_SEOL	shifted end of line
NCURSES_KEY_SEXIT	shifted exit
NCURSES_KEY_SFIND	shifted find
NCURSES_KEY_SHELP	shifted help
NCURSES_KEY_SHOME	shifted home
NCURSES_KEY_SIC	shifted input
NCURSES_KEY_SLEFT	shifted left arrow
NCURSES_KEY_SMESSAGE	shifted message
NCURSES_KEY_SMOVE	shifted move
NCURSES_KEY_SNEXT	shifted next
NCURSES_KEY_SOPTIONS	shifted options
NCURSES_KEY_SPREVIOUS	shifted previous
NCURSES_KEY_SPRINT	shifted print
NCURSES_KEY_SREDO	shifted redo
NCURSES_KEY_SREPLACE	shifted replace
NCURSES_KEY_SRIGHT	shifted right arrow
NCURSES_KEY_SRSUME	shifted resume

constant	meaning
NCURSES_KEY_SSAVE	shifted save
NCURSES_KEY_SSUSPEND	shifted suspend
NCURSES_KEY_UNDO	undo
NCURSES_KEY_MOUSE	mouse event has occurred
NCURSES_KEY_MAX	maximum key value

Mouse

Table 3. mouse constants

Constant	meaning
NCURSES_BUTTON1_RELEASED - NCURSES_BUTTON4_RELEASED	button (1-4) released
NCURSES_BUTTON1_PRESSED - NCURSES_BUTTON4_PRESSED	button (1-4) pressed
NCURSES_BUTTON1_CLICKED - NCURSES_BUTTON4_CLICKED	button (1-4) clicked
NCURSES_BUTTON1_DOUBLE_CLICKED - NCURSES_BUTTON4_DOUBLE_CLICKED	button (1-4) double clicked
NCURSES_BUTTON1_TRIPLE_CLICKED - NCURSES_BUTTON4_TRIPLE_CLICKED	button (1-4) triple clicked
NCURSES_BUTTON_CTRL	ctrl pressed during click
NCURSES_BUTTON_SHIFT	shift pressed during click
NCURSES_BUTTON_ALT	alt pressed during click
NCURSES_ALL_MOUSE_EVENTS	report all mouse events
NCURSES_REPORT_MOUSE_POSITION	report mouse position

ncurses_addch (PHP 4 >= 4.1.0)

Add character at current position and advance cursor

```
int ncurses_addch ( int ch) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_addchnstr (PHP 4 >= 4.2.0)

Add attributed string with specified length at current position

```
int ncurses_addchnstr ( string s, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_addchstr (PHP 4 >= 4.2.0)

Add attributed string at current position

```
int ncurses_addchstr ( string s) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_addnstr (PHP 4 >= 4.2.0)

Add string with specified length at current position

```
int ncurses_addnstr ( string s, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_addstr (PHP 4 >= 4.2.0)

Output text at current position

```
int ncurses_addstr ( string text) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_assume_default_colors (PHP 4 >= 4.2.0)

Define default colors for color 0

```
int ncurses_assume_default_colors ( int fg, int bg) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_attroff (PHP 4 >= 4.1.0)

Turn off the given attributes

```
int ncurses_attroff ( int attributes) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_attron (PHP 4 >= 4.1.0)

Turn on the given attributes

```
int ncurses_attron ( int attributes) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_attrset (PHP 4 >= 4.1.0)

Set given attributes

```
int ncurses_attrset ( int attributes) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_baudrate (PHP 4 >= 4.1.0)

Returns baudrate of terminal

```
int ncurses_baudrate ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_beep (PHP 4 >= 4.1.0)

Let the terminal beep

```
int ncurses_beep ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_beep() sends an audible alert (bell) and if its not possible flashes the screen. Returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_flash()`

ncurses_bkgd (PHP 4 >= 4.1.0)

Set background property for terminal screen

```
int ncurses_bkgd ( int attrchar) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_bkgdset (PHP 4 >= 4.1.0)

Control screen background

```
void ncurses_bkgdset ( int attrchar) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_border (PHP 4 >= 4.2.0)

Draw a border around the screen using attributed characters

```
int ncurses_border ( int left, int right, int top, int bottom, int tl_corner, int tr_corner, int bl_corner, int br_corner) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_can_change_color (PHP 4 >= 4.1.0)

Check if we can change terminals colors

```
bool ncurses_can_change_color ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The function **ncurses_can_change_color()** returns `TRUE` or `FALSE`, depending on whether the terminal has color capabilities and whether the programmer can change the colors.

ncurses_cbreak (PHP 4 >= 4.1.0)

Switch of input buffering

bool **ncurses_cbreak** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_cbreak() disables line buffering and character processing (interrupt and flow control characters are unaffected), making characters typed by the user immediately available to the program.

ncurses_cbreak() returns `TRUE` or `NCURSES_ERR` if any error occurred.

See also: `ncurses_nocbreak()`

ncurses_clear (PHP 4 >= 4.1.0)

Clear screen

bool **ncurses_clear** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_clear() clears the screen completely without setting blanks. Returns `FALSE` on success, otherwise `TRUE`.

Note: **ncurses_clear()** clears the screen without setting blanks, which have the current background rendition. To clear screen with blanks, use `ncurses_erase()`.

See also: `ncurses_erase()`

ncurses_clrtobot (PHP 4 >= 4.1.0)

Clear screen from current position to bottom

bool **ncurses_clrtobot** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_clrtoeol() erases all lines from cursor to end of screen and creates blanks. Blanks created by **ncurses_clrtoeol()** have the current background rendition. Returns `TRUE` if any error occurred, otherwise `FALSE`.

See also: `ncurses_clear()`, `ncurses_clrtoeol()`

ncurses_clrtoeol (PHP 4 >= 4.1.0)

Clear screen from current position to end of line

`bool ncurses_clrtoeol (void) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_clrtoeol() erases the current line from cursor position to the end. Blanks created by **ncurses_clrtoeol()** have the current background rendition. Returns `TRUE` if any error occurred, otherwise `FALSE`.

See also: `ncurses_clear()`, `ncurses_clrtoeol()`

ncurses_color_set (PHP 4 >= 4.1.0)

Set fore- and background color

`int ncurses_color_set (int pair) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_curs_set (PHP 4 >= 4.1.0)

Set cursor state

`int ncurses_curs_set (int visibility) \linebreak`**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_def_prog_mode (PHP 4 >= 4.1.0)

Saves terminals (program) mode

`bool ncurses_def_prog_mode (void) \linebreak`**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_def_prog_mode() saves the current terminal modes for program (in curses) for use by **ncurses_reset_prog_mode()**. Returns `FALSE` on success, otherwise `TRUE`.

See also: **ncurses_reset_prog_mode()**

ncurses_def_shell_mode (PHP 4 >= 4.1.0)

Saves terminals (shell) mode

`bool ncurses_def_shell_mode (void) \linebreak`**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_def_shell_mode() saves the current terminal modes for shell (not in curses) for use by **ncurses_reset_shell_mode()**. Returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_reset_shell_mode()`

`ncurses_define_key` (PHP 4 >= 4.2.0)

Define a keycode

```
int ncurses_define_key ( string definition, int keycode) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

`ncurses_delay_output` (PHP 4 >= 4.1.0)

Delay output on terminal using padding characters

```
int ncurses_delay_output ( int milliseconds) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

`ncurses_delch` (PHP 4 >= 4.1.0)

Delete character at current position, move rest of line left

```
bool ncurses_delch ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_delch() deletes the character under the cursor. All characters to the right of the cursor on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change. Returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_deleteln()`

ncurses_deleteln (PHP 4 >= 4.1.0)

Delete line at current position, move rest of screen up

bool **ncurses_deleteln** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_deleteln() deletes the current line under cursorposition. All lines below the current line are moved up one line. The bottom line of window is cleared. Cursor position does not change. Returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_delch()`

ncurses_delwin (PHP 4 >= 4.1.0)

Delete a ncurses window

int **ncurses_delwin** (resource window) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_douupdate (PHP 4 >= 4.1.0)

Write all prepared refreshes to terminal

bool **ncurses_douupdate** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_doupdate() compares the virtual screen to the physical screen and updates the physical screen. This way is more effective than using multiple refresh calls. Returns `FALSE` on success, `TRUE` if any error occurred.

ncurses_echo (PHP 4 >= 4.1.0)

Activate keyboard input echo

bool **ncurses_echo** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_echo() enables echo mode. All characters typed by user are echoed by `ncurses_getch()`. Returns `FALSE` on success, `TRUE` if any error occurred.

To disable echo mode use `ncurses_noecho()`.

ncurses_echochar (PHP 4 >= 4.1.0)

Single character output including refresh

int **ncurses_echochar** (int character) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_end (PHP 4 >= 4.1.0)

Stop using ncurses, clean up the screen

```
int ncurses_end ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_erase (PHP 4 >= 4.1.0)

Erase terminal screen

```
bool ncurses_erase ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_erase() fills the terminal screen with blanks. Created blanks have the current background rendition, set by `ncurses_bkgd()`. Returns `FALSE` on success, `TRUE` if any error occurred.

See also: `ncurses_bkgd()`, `ncurses_clear()`

ncurses_erasechar (PHP 4 >= 4.1.0)

Returns current erase character

```
string ncurses_erasechar ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_erasechar() returns the current erase char character.

See also: `ncurses_killchar()`

ncurses_filter (PHP 4 >= 4.1.0)

int **ncurses_filter** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_flash (PHP 4 >= 4.1.0)

Flash terminal screen (visual bell)

bool **ncurses_flash** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_flash() flashes the screen, and if its not possible, sends an audible alert (bell). Returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_beep()`

ncurses_flushinp (PHP 4 >= 4.1.0)

Flush keyboard input buffer

bool **ncurses_flushinp** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The **ncurses_flushinp()** throws away any typeahead that has been typed and has not yet been read by your program. Returns `FALSE` on success, otherwise `TRUE`.

ncurses_getch (PHP 4 >= 4.1.0)

Read a character from keyboard

```
int ncurses_getch ( void ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_getmouse (PHP 4 >= 4.2.0)

Reads mouse event

```
bool ncurses_getmouse ( array mevent ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_getmouse() reads mouse event out of queue. Function **ncurses_getmouse()** will return `FALSE` if a mouse event is actually visible in the given window, otherwise it will return `TRUE`. Event options will be delivered in parameter *mevent*, which has to be an array, passed by reference (see example below). On success an associative array with following keys will be delivered:

- "id" : Id to distinguish multiple devices
- "x" : screen relative x-position in character cells
- "y" : screen relative y-position in character cells
- "z" : currently not supported
- "mmask" : Mouse action

Example 1. ncurses_getmouse() example

```

switch (ncurses_getch){
  case NCURSES_KEY_MOUSE:
    if (!ncurses_getmouse(&$mevent)){
      if ($mevent["mmask"] & NCURSES_MOUSE_BUTTON1_PRESSED){
        $mouse_x = $mevent["x"]; // Save mouse position
        $mouse_y = $mevent["y"];
      }
    }
    break;

  default:
    ....
}

```

See also: `ncurses_ungetmouse()`

ncurses_halfdelay (PHP 4 >= 4.1.0)

Put terminal into halfdelay mode

```
int ncurses_halfdelay ( int tenth) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_has_colors (PHP 4 >= 4.1.0)

Check if terminal has colors

```
bool ncurses_has_colors ( void) \linebreak
```


Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_has_colors() returns `TRUE` or `FALSE` depending on whether the terminal has color capabilities.

See also: `ncurses_can_change_color()`

ncurses_has_ic (PHP 4 >= 4.1.0)

Check for insert- and delete-capabilities

```
bool ncurses_has_ic ( void ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_has_ic() checks terminal's insert- and delete capabilities. It returns `TRUE` when terminal has insert/delete-capabilities, otherwise `FALSE`.

See also: `ncurses_has_il()`

ncurses_has_il (PHP 4 >= 4.1.0)

Check for line insert- and delete-capabilities

```
bool ncurses_has_il ( void ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_has_il() checks terminal's insert- and delete-line capabilities. It returns `TRUE` when terminal has insert/delete-line capabilities, otherwise `FALSE`

See also: `ncurses_has_ic()`

ncurses_has_key (PHP 4 >= 4.1.0)

Check for presence of a function key on terminal keyboard

```
int ncurses_has_key ( int keycode) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_hline (PHP 4 >= 4.2.0)

Draw a horizontal line at current position using an attributed character and max. n characters long

```
int ncurses_hline ( int charattr, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_inch (PHP 4 >= 4.1.0)

Get character and attribute at current position

```
string ncurses_inch ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_inch() returns the character from the current position.

ncurses_init (PHP 4 >= 4.1.0)

Initialize ncurses

int **ncurses_init** (void) \linebreak**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_init_color (PHP 4 >= 4.2.0)

Set new RGB value for color

int **ncurses_init_color** (int color, int r, int g, int b) \linebreak**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_init_pair (PHP 4 >= 4.1.0)

Allocate a color pair

int **ncurses_init_pair** (int pair, int fg, int bg) \linebreak**Warning**

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_insch (PHP 4 >= 4.1.0)

Insert character moving rest of line including character at current position

```
int ncurses_insch ( int character) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_insdelln (PHP 4 >= 4.1.0)

Insert lines before current line scrolling down (negative numbers delete and scroll up)

```
int ncurses_insdelln ( int count) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_insertln (PHP 4 >= 4.1.0)

Insert a line, move rest of screen down

```
bool ncurses_insertln ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_insertln() inserts a new line above the current line. The bottom line will be lost.

ncurses_Insstr (PHP 4 >= 4.2.0)

Insert string at current position, moving rest of line right

```
int ncurses_Insstr ( string text) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_instr (PHP 4 >= 4.2.0)

Reads string from terminal screen

```
int ncurses_instr ( string buffer) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_instr() returns the number of characters read from the current character position until end of line. *buffer* contains the characters. Attributes are stripped from the characters.

ncurses_Isendwin (PHP 4 >= 4.1.0)

Ncurses is in endwin mode, normal screen output may be performed

```
bool ncurses_Isendwin ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_Isendwin() returns `TRUE`, if **ncurses_endwin()** has been called without any subsequent calls to `ncurses_wrefresh()`, otherwise `FALSE`.

See also: **ncurses_endwin()** **ncurses_wrefresh()**

ncurses_keyok (PHP 4 >= 4.2.0)

Enable or disable a keycode

```
int ncurses_keyok ( int keycode, bool enable) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_killchar (PHP 4 >= 4.1.0)

Returns current line kill character

```
bool ncurses_killchar ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_killchar() returns the current line kill character.

See also: `ncurses_erasechar()`

ncurses_longname (PHP 4 >= 4.2.0)

Returns terminals description

```
string ncurses_longname ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_longname() returns a verbose description of the terminal. The description is truncated to 128 characters. On Error **ncurses_longname()** returns NULL.

See also: `ncurses_termname()`

ncurses_mouseinterval (PHP 4 >= 4.1.0)

Set timeout for mouse button clicks

`int ncurses_mouseinterval (int milliseconds) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mousemask (PHP 4 >= 4.2.0)

Sets mouse options

`int ncurses_mousemask (int newmask, int oldmask) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Function `ncurses_mousemask()` will set mouse events to be reported. By default no mouse events will be reported. The function `ncurses_mousemask()` will return a mask to indicated which of the in parameter *newmask* specified mouse events can be reported. On complete failure, it returns 0. In parameter *oldmask*, which is passed by reference `ncurses_mousemask()` returns the previous value of mouse event mask. Mouse events are represented bei `NCURSES_KEY_MOUSE` in the `ncurses_wgetch()` input stream. To read the event data and pop the event of of queue, call `ncurses_getmouse()`.

As a side effect, setting a zero mousemask in *newmask* turns off the mouse pointer. Setting a non zero value turns mouse pointer on.

mouse mask options can be set with the following predefined constants:

- `NCURSES_BUTTON1_PRESSED`
- `NCURSES_BUTTON1_RELEASED`
- `NCURSES_BUTTON1_CLICKED`

- NCURSES_BUTTON1_DOUBLE_CLICKED
- NCURSES_BUTTON1_TRIPLE_CLICKED
- NCURSES_BUTTON2_PRESSED
- NCURSES_BUTTON2_RELEASED
- NCURSES_BUTTON2_CLICKED
- NCURSES_BUTTON2_DOUBLE_CLICKED
- NCURSES_BUTTON2_TRIPLE_CLICKED
- NCURSES_BUTTON3_PRESSED
- NCURSES_BUTTON3_RELEASED
- NCURSES_BUTTON3_CLICKED
- NCURSES_BUTTON3_DOUBLE_CLICKED
- NCURSES_BUTTON3_TRIPLE_CLICKED
- NCURSES_BUTTON4_PRESSED
- NCURSES_BUTTON4_RELEASED
- NCURSES_BUTTON4_CLICKED
- NCURSES_BUTTON4_DOUBLE_CLICKED
- NCURSES_BUTTON4_TRIPLE_CLICKED
- NCURSES_BUTTON_SHIFT>
- NCURSES_BUTTON_CTRL
- NCURSES_BUTTON_ALT
- NCURSES_ALL_MOUSE_EVENTS
- NCURSES_REPORT_MOUSE_POSITION

See also: `ncurses_getmouse()`, `ncurses_ungetmouse()` **`ncurses_getch()`**

Example 1. `ncurses_mousemask()` example

```
$newmask = NCURSES_BUTTON1_CLICKED + NCURSES_BUTTON1_RELEASED;  
$mask = ncurses_mousemask($newmask, &$oldmask);  
if ($mask & $newmask){  
    printf ("All specified mouse options will be supported\n");  
}
```


ncurses_move (PHP 4 >= 4.1.0)

Move output position

```
int ncurses_move ( int y, int x) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvaddch (PHP 4 >= 4.2.0)

Move current position and add character

```
int ncurses_mvaddch ( int y, int x, int c) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvaddchnstr (PHP 4 >= 4.2.0)

Move position and add attributed string with specified length

```
int ncurses_mvaddchnstr ( int y, int x, string s, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvaddchstr (PHP 4 >= 4.2.0)

Move position and add attributed string

```
int ncurses_mvaddchstr ( int y, int x, string s ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvaddnstr (PHP 4 >= 4.2.0)

Move position and add string with specified length

```
int ncurses_mvaddnstr ( int y, int x, string s, int n ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvaddstr (PHP 4 >= 4.2.0)

Move position and add string

```
int ncurses_mvaddstr ( int y, int x, string s ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvcur (PHP 4 >= 4.2.0)

Move cursor immediately

```
int ncurses_mvcur ( int old_y, int old_x, int new_y, int new_x) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvdelch (PHP 4 >= 4.2.0)

Move position and delete character, shift rest of line left

```
int ncurses_mvdelch ( int y, int x) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvgetch (PHP 4 >= 4.2.0)

Move position and get character at new position

```
int ncurses_mvgetch ( int y, int x) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvhline (PHP 4 >= 4.2.0)

Set new position and draw a horizontal line using an attributed character and max. n characters long

```
int ncurses_mvhline ( int y, int x, int attrchar, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvinch (PHP 4 >= 4.2.0)

Move position and get attributed character at new position

```
int ncurses_mvinch ( int y, int x) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvvline (unknown)

Set new position and draw a vertical line using an attributed character and max. n characters long

```
int ncurses_mvvline ( int y, int x, int attrchar, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_mvwaddstr (PHP 4 >= 4.2.0)

Add string at new position in window

int **ncurses_mvwaddstr** (resource window, int y, int x, string text) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_napms (PHP 4 >= 4.1.0)

Sleep

int **ncurses_napms** (int milliseconds) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_newwin (PHP 4 >= 4.1.0)

Create a new window

int **ncurses_newwin** (int rows, int cols, int y, int x) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_nl (PHP 4 >= 4.1.0)

Translate newline and carriage return / line feed

bool **ncurses_nl** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_nocbreak (PHP 4 >= 4.1.0)

Switch terminal to cooked mode

bool **ncurses_nocbreak** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_nocbreak() routine returns terminal to normal (cooked) mode. Initially the terminal may or may not in cbreak mode as the mode is inherited. Therefore a program should call **ncurses_cbreak()** and **ncurses_nocbreak()** explicitly. Returns `TRUE` if any error occurred, otherwise `FALSE`.

See also: **ncurses_cbreak()**

ncurses_noecho (PHP 4 >= 4.1.0)

Switch off keyboard input echo

bool **ncurses_noecho** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_noecho() prevents echoing of user typed characters. Returns `TRUE` if any error occurred, otherwise `FALSE`.

See also: `ncurses_echo()`, `ncurses_getch()`

ncurses_nonl (PHP 4 >= 4.1.0)

Do not translate newline and carriage return / line feed

bool **ncurses_nonl** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_noqiflush (PHP 4 >= 4.1.0)

Do not flush on signal characters

int **ncurses_noqiflush** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_noraw (PHP 4 >= 4.1.0)

Switch terminal out of raw mode

bool **ncurses_noraw** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_noraw() switches the terminal out of raw mode. Raw mode is similar to cbreak mode, in that characters typed are immediately passed through to the user program. The differences that are that in raw mode, the interrupt, quit, suspend and flow control characters are all passed through uninterpreted, instead of generating a signal. Returns **TRUE** if any error occurred, otherwise **FALSE**.

See also: `ncurses_raw()`, `ncurses_cbreak()`, `ncurses_nocbreak()`

ncurses_putp (PHP 4 >= 4.2.0)

int **ncurses_putp** (string text) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_qiflush (PHP 4 >= 4.1.0)

Flush on signal characters

int **ncurses_qiflush** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_raw (PHP 4 >= 4.1.0)

Switch terminal into raw mode

```
bool ncurses_raw ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_raw() places the terminal in raw mode. Raw mode is similar to cbreak mode, in that characters typed are immediately passed through to the user program. The differences that are that in raw mode, the interrupt, quit, suspend and flow control characters are all passed through uninterpreted, instead of generating a signal. Returns *TRUE* if any error occurred, otherwise *FALSE*.

See also: `ncurses_noraw()`, `ncurses_cbreak()`, `ncurses_nocbreak()`

ncurses_refresh (PHP 4 >= 4.1.0)

Refresh screen

```
int ncurses_refresh ( int ch) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_resetty (PHP 4 >= 4.1.0)

Restores saved terminal state

```
bool ncurses_resetty ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Function **ncurses_resetty()** restores the terminal state, which was previously saved by calling **ncurses_savetty()**. This function always returns `FALSE`.

See also: **ncurses_savetty()**

ncurses_savetty (PHP 4 >= 4.1.0)

Saves terminal state

bool **ncurses_savetty** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Function **ncurses_savetty()** saves the current terminal state. The saved terminal state can be restored with function **ncurses_resetty()**. **ncurses_savetty()** always returns `FALSE`.

See also: **ncurses_resetty()**

ncurses_scr_dump (PHP 4 >= 4.2.0)

Dump screen content to file

int **ncurses_scr_dump** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_scr_init (PHP 4 >= 4.2.0)

Initialize screen from file dump

int **ncurses_scr_init** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_scr_restore (PHP 4 >= 4.2.0)

Restore screen from file dump

int **ncurses_scr_restore** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_scr_set (PHP 4 >= 4.2.0)

Inherit screen from file dump

int **ncurses_scr_set** (string filename) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_scrll (PHP 4 >= 4.1.0)

Scroll window content up or down without changing current position

int **ncurses_scrll** (int count) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_attr (PHP 4 >= 4.1.0)

Returns current soft label key attribute

bool **ncurses_slk_attr** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_slk_attr() returns the current soft label key attribute. On error returns `TRUE`, otherwise `FALSE`.

ncurses_slk_attroff (PHP 4 >= 4.1.0)

int **ncurses_slk_attroff** (int intarg) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_attron (PHP 4 >= 4.1.0)

int **ncurses_slk_attron** (int intarg) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_attrset (PHP 4 >= 4.1.0)

int **ncurses_slk_attrset** (int intarg) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_clear (PHP 4 >= 4.1.0)

Clears soft labels from screen

bool **ncurses_slk_clear** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The function **ncurses_slk_clear()** clears soft label keys from screen. Returns `TRUE` on error, otherwise `FALSE`.

ncurses_slk_color (PHP 4 >= 4.1.0)

Sets color for soft label keys

int **ncurses_slk_color** (int intarg) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_init (PHP 4 >= 4.1.0)

Initializes soft label key functions

bool **ncurses_slk_init** (int *format*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Function **ncurses_slk_init()** must be called before **ncurses_initscr()** or **ncurses_newterm()** is called. If **ncurses_initscr()** eventually uses a line from `stdscr` to emulate the soft labels, then *format* determines how the labels are arranged of the screen. Setting *format* to 0 indicates a 3-2-3 arrangement of the labels, 1 indicates a 4-4 arrangement and 2 indicates the PC like 4-4-4 mode, but in addition an index line will be created.

ncurses_slk_noutrefresh (PHP 4 >= 4.1.0)

Copies soft label keys to virtual screen

bool **ncurses_slk_noutrefresh** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_slk_refresh (PHP 4 >= 4.1.0)

Copies soft label keys to screen

```
bool ncurses_slk_refresh ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_slk_refresh() copies soft label keys from virtual screen to physical screen. Returns `TRUE` on error, otherwise `FALSE`.

ncurses_slk_restore (PHP 4 >= 4.1.0)

Restores soft label keys

```
bool ncurses_slk_restore ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The function **ncurses_slk_restore()** restores the soft label keys after `ncurses_slk_clear()` has been performed.

ncurses_slk_touch (PHP 4 >= 4.1.0)

Forces output when `ncurses_slk_noutrefresh` is performed

```
bool ncurses_slk_touch ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The **ncurses_slk_touch()** function forces all the soft labels to be output the next time a `ncurses_slk_noutrefresh()` is performed.

ncurses_standend (PHP 4 >= 4.1.0)

Stop using 'standout' attribute

```
int ncurses_standend ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_standout (PHP 4 >= 4.1.0)

Start using 'standout' attribute

```
int ncurses_standout ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_start_color (PHP 4 >= 4.1.0)

Start using colors

```
int ncurses_start_color ( void) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_termattrs (PHP 4 >= 4.1.0)

Returns a logical OR of all attribute flags supported by terminal

bool **ncurses_termattrs** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_termname (PHP 4 >= 4.2.0)

Returns terminals (short)-name

string **ncurses_termname** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

ncurses_termname() returns terminals shortname. The shortname is truncated to 14 characters. On error **ncurses_termname**() returns NULL.

See also: **ncurses_longname**()

ncurses_timeout (PHP 4 >= 4.1.0)

Set timeout for special key sequences

void **ncurses_timeout** (int millisec) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_typeahead (PHP 4 >= 4.1.0)

Specify different filedescriptor for typeahead checking

```
int ncurses_typeahead ( int fd) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_ungetch (PHP 4 >= 4.1.0)

Put a character back into the input stream

```
int ncurses_ungetch ( int keycode) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_ungetmouse (PHP 4 >= 4.2.0)

Pushes mouse event to queue

```
bool ncurses_ungetmouse ( array mevent) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`ncurses_getmouse()` pushes a `KEY_MOUSE` event onto the unput queue and associates with this event the given state `sata` and screen-relative character cell coordinates, specified in `mevent`. Event options will be specified in associative array `mevent`:

- "id" : Id to distinguish multiple devices
- "x" : screen relative x-position in character cells
- "y" : screen relative y-position in character cells
- "z" : currently not supported
- "mmask" : Mouse action

ncurses_ungetmouse() returns `FALSE` on success, otherwise `TRUE`.

See also: `ncurses_getmouse()`

ncurses_use_default_colors (PHP 4 >= 4.1.0)

Assign terminal default colors to color id -1

`bool ncurses_use_default_colors (void) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_use_env (PHP 4 >= 4.1.0)

Control use of environment information about terminal size

`void ncurses_use_env (bool flag) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_use_extended_names (PHP 4 >= 4.1.0)

Control use of extended names in terminfo descriptions

```
int ncurses_use_extended_names ( bool flag) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_vidattr (PHP 4 >= 4.1.0)

```
int ncurses_vidattr ( int intarg) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_vline (PHP 4 >= 4.2.0)

Draw a vertical line at current position using an attributed character and max. n characters long

```
int ncurses_vline ( int charattr, int n) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

ncurses_wrefresh (PHP 4 >= 4.2.0)

Refresh window on terminal screen

```
int ncurses_wrefresh ( resource window) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

undocumented

LXVII. Lotus Notes functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

notes_body (PHP 4 >= 4.0.5)

Open the message `msg_number` in the specified mailbox on the specified server (leave serv
array **notes_body** (string server, string mailbox, int msg_number) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_copy_db (PHP 4 >= 4.0.5)

Create a note using form `form_name`

string **notes_copy_db** (string from_database_name, string to_database_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_create_db (PHP 4 >= 4.0.5)

Create a Lotus Notes database

bool **notes_create_db** (string database_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_create_note (PHP 4 >= 4.0.5)

Create a note using form form_name

```
string notes_create_note ( string database_name, string form_name) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_drop_db (PHP 4 >= 4.0.5)

Drop a Lotus Notes database

```
bool notes_drop_db ( string database_name) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_find_note (PHP 4 >= 4.0.5)

Returns a note id found in database_name. Specify the name of the note. Leaving type bla

bool **notes_find_note** (string database_name, string name [, string type]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_header_info (PHP 4 >= 4.0.5)

Open the message msg_number in the specified mailbox on the specified server (leave serv

object **notes_header_info** (string server, string mailbox, int msg_number) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_list_msgs (PHP 4 >= 4.0.5)

Returns the notes from a selected database_name

bool **notes_list_msgs** (string db) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_mark_read (PHP 4 >= 4.0.5)

Mark a note_id as read for the User user_name

string **notes_mark_read** (string database_name, string user_name, string note_id) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_mark_unread (PHP 4 >= 4.0.5)

Mark a note_id as unread for the User user_name

string **notes_mark_unread** (string database_name, string user_name, string note_id) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_nav_create (PHP 4 >= 4.0.5)

Create a navigator name, in database_name

bool **notes_nav_create** (string database_name, string name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_search (PHP 4 >= 4.0.5)

Find notes that match keywords in database_name

string **notes_search** (string database_name, string keywords) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_unread (PHP 4 >= 4.0.5)

Returns the unread note id's for the current User user_name

string **notes_unread** (string database_name, string user_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

notes_version (PHP 4 >= 4.0.5)

Get the version Lotus Notes

string **notes_version** (string database_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

LXVIII. 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.

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. The exception to this is iODBC. Building PHP with iODBC support enables you to use any ODBC-compliant drivers with your PHP applications. iODBC is maintained by OpenLink Software (<http://www.openlinksw.com/>). More information on iODBC, as well as a HOWTO, is available at www.iodbc.org (<http://www.iodbc.org/>).

Requirements

The following databases are supported by the Unified ODBC functions: Adabas D (<http://www.software-ag.com/adabasd/>), 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/>). To access these databases you need to have the required libraries installed.

Installation

Please see the Installation on Unix Systems chapter for more information about configuring PHP with these databases.

Runtime Configuration

The behaviour of the ODBC functions is affected by settings in the global configuration file `php.ini`.

Table 1. Unified ODBC Configuration Options

Name	Default	Changeable
<code>odbc.default_db *</code>	NULL	PHP_INI_ALL
<code>odbc.default_user *</code>	NULL	PHP_INI_ALL
<code>odbc.default_pw *</code>	NULL	PHP_INI_ALL
<code>odbc.allow_persistent</code>	"1"	PHP_INI_SYSTEM
<code>odbc.check_persistent</code>	"1"	PHP_INI_SYSTEM
<code>odbc.max_persistent</code>	"-1"	PHP_INI_SYSTEM
<code>odbc.max_links</code>	"-1"	PHP_INI_SYSTEM

Name	Default	Changeable
odbc.defaultlrl	"4096"	PHP_INI_ALL
odbc.defaultbinmode	"1"	PHP_INI_ALL

Note: Entries marked with * are not implemented yet.

For further details and definition of the PHP_INI_* constants see ini_set().

Here is a short explanation of the configuration directives.

odbc.default_db string

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

odbc.default_user string

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

odbc.default_pw string

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

odbc.allow_persistent boolean

Whether to allow persistent ODBC connections.

odbc.check_persistent boolean

Check that a connection is still valid before reuse.

odbc.max_persistent integer

The maximum number of persistent ODBC connections per process.

odbc.max_links integer

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

odbc.defaultlrl integer

Handling of LONG fields. Specifies the number of bytes returned to variables.

odbc.defaultbinmode integer

Handling of binary data.

Resource types

This extension does not define any resource types.

Predefined constants

This extension does not define any constants.

odbc_autocommit (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Toggle autocommit behaviour

```
bool odbc_autocommit ( resource connection_id [, bool OnOff]) \linebreak
```

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 >= 4.0.0)

Handling of binary column data

```
int odbc_binmode ( resource result_id, int mode) \linebreak
```

(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 >= 4.0.0)

Close an ODBC connection

```
void odbc_close ( resource connection_id) \linebreak
```

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 >= 4.0.0)

Close all ODBC connections

```
void odbc_close_all ( void) \linebreak
```

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_columnprivileges (PHP 4 >= 4.0.0)

Returns a result identifier that can be used to fetch a list of columns and associated privileges

`int odbc_columnprivileges (resource connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]]) \linebreak`

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_columns (PHP 4 >= 4.0.0)

Lists the column names in specified tables. Returns a result identifier containing the information.

`int odbc_columns (resource connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]]) \linebreak`

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_commit (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Commit an ODBC transaction

bool **odbc_commit** (resource connection_id) \linebreak

Returns: TRUE on success, FALSE on failure. All pending transactions on *connection_id* are committed.

odbc_connect (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Connect to a datasource

resource **odbc_connect** (string dsn, string user, string password [, int cursor_type]) \linebreak

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 >= 4.0.0)

Get cursorname

string **odbc_cursor** (resource result_id) \linebreak

`odbc_cursor` will return a cursorname for the given `result_id`.

odbc_do (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Synonym for `odbc_exec()`

resource **odbc_do** (resource conn_id, string query) \linebreak

`odbc_do()` will execute a query on the given connection.

odbc_error (PHP 4 >= 4.0.5)

Get the last error code

string **odbc_error** ([resource connection_id]) \linebreak

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 (PHP 4 >= 4.0.5)

Get the last error message

string **odbc_errormsg** ([resource connection_id]) \linebreak

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 >= 4.0.0)

Prepare and execute a SQL statement

resource **odbc_exec** (resource connection_id, string query_string) \linebreak

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 >= 4.0.0)

Execute a prepared statement

resource **odbc_execute** (resource result_id [, array parameters_array]) \linebreak

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.

Parameters in *parameter_array* will be substituted for placeholders in the prepared statement in order.

Any parameters in *parameter_array* which start and end with single quotes will be taken as the name of a file to read and send to the database server as the data for the appropriate placeholder.

Note: As of PHP 4.1.1, this file reading functionality has the following restrictions:

- File reading is *not* subject to any safe mode or safe mode restrictions. This is fixed in PHP 4.2.0.
- Remote files are not supported.
- If you wish to store a string which actually begins and ends with single quotes, you must escape them or add a space or other non-single-quote character to the beginning or end of the parameter, which will prevent the parameter's being taken as a file name. If this is not an option, then you must use another mechanism to store the string, such as executing the query directly with `odbc_exec()`.

odbc_fetch_array (PHP 4 >= 4.0.2)

Fetch a result row as an associative array

array **odbc_fetch_array** (resource result [, int rownumber]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

odbc_fetch_into (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Fetch one result row into array

bool **odbc_fetch_into** (resource result_id [, int rownumber, array result_array]) \linebreak resource **odbc_fetch_into** (resource result_id, array result_array [, int rownumber]) \linebreak

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.

As of PHP 4.0.5 the *result_array* does not need to be passed by reference any longer.

As of PHP 4.0.6 the *rownumber* cannot be passed as a constant, but rather as a variable.

As of PHP 4.2.0 the *result_array* and *rownumber* have been swapped. This allows the *rownumber* to be a constant again. This change will also be the last one to this function.

Example 1. odbc_fetch_into() pre 4.0.6 example

```
$rc = odbc_fetch_into($res_id, $my_array);
```

or

```
$rc = odbc_fetch_into($res_id, $row, $my_array);
```

```
$rc = odbc_fetch_into($res_id, 1, $my_array);
```

Example 2. odbc_fetch_into() 4.0.6 example

```
$rc = odbc_fetch_into($res_id, $my_array);
```

or

```
$row = 1;  
$rc = odbc_fetch_into($res_id, $row, $my_array);
```

Example 3. odbc_fetch_into() 4.2.0 example

```
$rc = odbc_fetch_into($res_id, $my_array);
```

or

```
$rc = odbc_fetch_into($res_id, $my_array, 2);
```

odbc_fetch_object (PHP 4 >= 4.0.2)

Fetch a result row as an object

object **odbc_fetch_object** (resource result [, int rownumber]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

odbc_fetch_row (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Fetch a row

bool **odbc_fetch_row** (resource result_id [, int row_number]) \linebreak

If **odbc_fetch_row()** was succesful (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_len (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the length (precision) of a field

```
int odbc_field_len ( resource result_id, int field_number) \linebreak
```

odbc_field_len() will return the length of the field referenced 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_name (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get the columnname

```
string odbc_field_name ( resource result_id, int field_number) \linebreak
```

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 >= 4.0.0)

Return column number

```
int odbc_field_num ( resource result_id, string field_name) \linebreak
```

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_precision (PHP 4 >= 4.0.0)

Synonym for `odbc_field_len()`

string **odbc_field_precision** (resource result_id, int field_number) \linebreak

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** (resource result_id, int field_number) \linebreak

`odbc_field_precision()` will return the scale of the field referenced by number in the given ODBC result identifier.

odbc_field_type (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Datatype of a field

string **odbc_field_type** (resource result_id, int field_number) \linebreak

odbc_field_type() will return the SQL type of the field referenced by number in the given ODBC result identifier. Field numbering starts at 1.

odbc_foreignkeys (PHP 4 >= 4.0.0)

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

resource **odbc_foreignkeys** (resource connection_id, string pk_qualifier, string pk_owner, string pk_table, string fk_qualifier, string fk_owner, string fk_table) \linebreak

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_free_result (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Free resources associated with a result

```
bool odbc_free_result ( resource result_id ) \linebreak
```

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_gettypeinfo (PHP 4 >= 4.0.0)

Returns a result identifier containing information about data types supported by the data source.

int **odbc_gettypeinfo** (resource connection_id [, int data_type]) \linebreak

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_longreadlen (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Handling of LONG columns

int **odbc_longreadlen** (resource result_id, int length) \linebreak

(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_next_result (PHP 4 >= 4.0.5)

Checks if multiple results are available

```
bool odbc_next_result ( resource result_id) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

odbc_num_fields (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Number of columns in a result

```
int odbc_num_fields ( resource result_id) \linebreak
```

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_num_rows (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Number of rows in a result

```
int odbc_num_rows ( resource result_id) \linebreak
```

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_pconnect (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Open a persistent database connection

```
int odbc_pconnect ( string dsn, string user, string password [, int cursor_type]) \linebreak
```

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 >= 4.0.0)

Prepares a statement for execution

resource **odbc_prepare** (resource connection_id, string query_string) \linebreak

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_primarykeys (PHP 4 >= 4.0.0)

Returns a result identifier that can be used to fetch the column names that comprise the primary key for a table

resource **odbc_primarykeys** (resource connection_id, string qualifier, string owner, string table) \linebreak

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_procedurecolumns (PHP 4 >= 4.0.0)

Retrieve information about parameters to procedures

resource **odbc_procedurecolumns** (resource connection_id [, string qualifier [, string owner [, string proc [, string column]]]]) \linebreak

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_procedures (PHP 4 >= 4.0.0)

Get the list of procedures stored in a specific data source. Returns a result identifier containing the information.

resource **odbc_procedures** (resource connection_id [, string qualifier [, string owner [, string name]]) \linebreak

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_result (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get result data

string **odbc_result** (resource result_id, mixed field) \linebreak

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 >= 4.0.0)

Print result as HTML table

`int odbc_result_all (resource result_id [, string format]) \linebreak`

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 >= 4.0.0)

Rollback a transaction

`int odbc_rollback (resource connection_id) \linebreak`

Rolls back all pending statements on *connection_id*. Returns `TRUE` on success, `FALSE` on failure.

odbc_setoption (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Adjust ODBC settings. Returns `FALSE` if an error occurs, otherwise `TRUE`.

`int odbc_setoption (resource id, int function, int option, int param) \linebreak`

This function allows fiddling with the ODBC options for a particular connection or query result. It was written to help find work arounds 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_specialcolumns (PHP 4 >= 4.0.0)

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

resource **odbc_specialcolumns** (resource connection_id, int type, string qualifier, string owner, string table, int scope, int nullable) \linebreak

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 >= 4.0.0)

Retrieve statistics about a table

resource **odbc_statistics** (resource connection_id, string qualifier, string owner, string table_name, int unique, int accuracy) \linebreak

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`.

odbc_tableprivileges (PHP 4 >= 4.0.0)

Lists tables and the privileges associated with each table

int **odbc_tableprivileges** (resource connection_id [, string qualifier [, string owner [, string name]]]) \linebreak

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_tables (PHP 3 >= 3.0.17, PHP 4 >= 4.0.0)

Get the list of table names stored in a specific data source. Returns a result identifier containing the information.

```
int odbc_tables ( resource connection_id [, string qualifier [, string owner [, string name [, string types]]]])
\linebreak
```

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.

LXIX. 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.

If your webserver doesn't start or crashes at startup: Check that Apache is linked with the pthread library:

```
# ldd /www/apache/bin/httpd
  libpthread.so.0 => /lib/libpthread.so.0 (0x4001c000)
  libm.so.6 => /lib/libm.so.6 (0x4002f000)
  libcrypt.so.1 => /lib/libcrypt.so.1 (0x4004c000)
  libdl.so.2 => /lib/libdl.so.2 (0x4007a000)
  libc.so.6 => /lib/libc.so.6 (0x4007e000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

If the libpthread is not listed you have to reinstall Apache:

```
# cd /usr/src/apache_1.3.xx
# make clean
# LIBS=-lpthread ./config.status
# make
# make install
```

Example 1. OCI Hints

```

<?php
// by sergo@bacup.ru

// Use option: OCI_DEFAULT for execute command to delay execution
OCIExecute($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);
OCIExecute($stmt, OCI_DEFAULT);
$clob->save ("some text");
OCICommit($conn);

?>

```

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 );  
OCIExecute ( $sth );
```

?>

OCIBindByName (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Bind a PHP variable to an Oracle Placeholder

int **OCIBindByName** (int stmt, string ph_name, mixed & variable, int length [, int type]) \linebreak

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($data)) {
    OCIExecute($stmt);
    OCIExecute($update);
}

$rowid->free();
```

```

OCIFreeStatement($update);
OCIFreeStatement($stmt);

$stmt = OCIParse($conn,"select * from emp where empno in (1111,2222,3333)");
OCIExecute($stmt);
while (OCIFetchInto($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)");
OCIExecute($stmt);
OCIFreeStatement($stmt);

OCILogout($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 quotations from those added by intention.

OCICancel (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Cancel reading from cursor

```
int OCICancel ( int stmt) \linebreak
```

If you do not want read more data from a cursor, then call **OCICancel()**.

OCICollAppend (PHP 4 >= 4.0.6)

Coming soon

```
string OCICollAppend ( object collection, object object) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

OCICollAssign (PHP 4 >= 4.0.6)

Coming soon

string **OCICollAssign** (object collection, object object) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICollAssignElem (PHP 4 >= 4.0.6)

Coming soon

string **OCICollAssignElem** (object collection, string ndx, string val) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICollGetElem (PHP 4 >= 4.0.6)

Coming soon

string **OCICollGetElem** (object collection, string ndx) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICollMax (PHP 4 >= 4.0.6)

Coming soon

string **OCICollMax** (object collection) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICollSize (PHP 4 >= 4.0.6)

Coming soon

string **OCICollSize** (object collection) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICollTrim (PHP 4 >= 4.0.6)

Coming soon

string **OCICollTrim** (object collection, int num) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCIColumnIsNULL (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Test whether a result column is NULL

int **OCIColumnIsNULL** (int stmt, mixed column) \linebreak

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 >= 4.0.0)

Returns the name of a column

string **OCIColumnName** (int stmt, int col) \linebreak

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");
OCIExecute($stmt);
print "<TABLE BORDER=\\"1\">";
print "<TR>";
print "<TH>Name</TH>";
print "<TH>Type</TH>";
print "<TH>Length</TH>";
print "</TR>";
$numcols = OCINumCols($stmt);
for ( $i = 1; $i <= $numcols; $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>\n";
OCIFreeStatement($stmt);
OCILogoff($conn);
print "</PRE>";
print "</HTML>\n";
?>
```

See also `OCINumCols()`, `OCIColumnType()`, and `OCIColumnSize()`.

OCIColumnPrecision (PHP 4 >= 4.0.0)

Coming soon

int `OCIColumnPrecision` (int stmt, int col) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCIColumnScale (PHP 4 >= 4.0.0)

Coming soon

int `OCIColumnScale` (int stmt, int col) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCIColumnSize (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Return result column size

int `OCIColumnSize` (int stmt, mixed column) \linebreak

`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");
```

```

OCIExecute($stmt);
print "<TABLE BORDER=\"1\">";
print "<TR>";
print "<TH>Name</TH>";
print "<TH>Type</TH>";
print "<TH>Length</TH>";
print "</TR>";
$numcols = OCINumCols($stmt);
for ( $i = 1; $i <= $numcols; $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 >= 4.0.0)

Returns the data type of a column

mixed **OCIColumnType** (int stmt, int col) \linebreak

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");
OCIExecute($stmt);
print "<TABLE BORDER=\"1\">";
print "<TR>";

```

```

print "<TH>Name</TH>";
print "<TH>Type</TH>";
print "<TH>Length</TH>";
print "</TR>";
$numcols = OCINumCols($stmt);
for ( $i = 1; $i <= $numcols; $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>\n";
OCIFreeStatement($stmt);
OCILogoff($conn);
print "</PRE>";
print "</HTML>\n";
?>

```

See also `OCINumCols()`, `OCIColumnName()`, and `OCIColumnSize()`.

OCIColumnTypeRaw (PHP 4 >= 4.0.0)

Coming soon

mixed `OCIColumnTypeRaw` (int stmt, int col) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCICommit (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Commits outstanding transactions

int `OCICommit` (int connection) \linebreak

`OCICommit()` commits all outstanding statements for Oracle connection *connection*.

OCIDefineByName (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Use a PHP variable for the define-step during a SELECT

int **OCIDefineByName** (int stmt, string Column-Name, mixed variable [, int type]) \linebreak

OCIDefineByName() binds PHP variables for fetches of SQL-Columns. Be careful that Oracle uses ALL-UPPERCASE column-names, whereby in your select you can also write lowercase.

OCIDefineByName() expects the *Column-Name* to be in uppercase. If you define a variable that doesn't exist in your 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
/* OCIDefineByName 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);

OCIExecute($stmt);

while (OCIFetch($stmt)) {
    echo "empno:". $empno. "\n";
    echo "ename:". $ename. "\n";
}

OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIError (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Return the last error of stmt|conn|global

array **OCIError** ([int stmt|conn|global]) \linebreak

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.

OCIExecute (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Execute a statement

int **OCIExecute** (int statement [, int mode]) \linebreak

OCIExecute() 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.

Returns `TRUE` on success, `FALSE` on failure.

OCIFetch (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Fetches the next row into result-buffer

int **OCIFetch** (int statement) \linebreak

OCIFetch() fetches the next row (for `SELECT` statements) into the internal result-buffer.

OCIFetchInto (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Fetches the next row into result-array

int **OCIFetchInto** (int stmt, array & result [, int mode]) \linebreak

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 zero-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 zero. (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 >= 4.0.0)

Fetch all rows of result data into an array

int **OCIFetchStatement** (int stmt, array & variable) \linebreak

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");

OCIExecute($stmt);

$rows = OCIFetchStatement($stmt,$results);
if ( $rows > 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 < $rows; $i++ ) {
        reset($results);
        print "<TR>\n";
        while ( $column = each($results) ) {
            $data = $column['value'];
            print "<TD>$data[$i]</TD>\n";
        }
        print "</TR>\n";
    }
    print "</TABLE>\n";
} else {
    echo "No data found<BR>\n";
}
print "$rows Records Selected<BR>\n";

OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIFreeCollection (PHP 4 >= 4.1.0)

Coming soon

string **OCIFreeCollection** (object lob) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCIFreeCursor (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Free all resources associated with a cursor

int **OCIFreeCursor** (int stmt) \linebreak

OCIFreeCursor() returns **TRUE** if successful, or **FALSE** if unsuccessful.

OCIFreeDesc (PHP 4 >= 4.0.0)

Deletes a large object descriptor

int **OCIFreeDesc** (object lob) \linebreak

OCIFreeDesc() returns **TRUE** if successful, or **FALSE** if unsuccessful.

OCIFreeStatement (PHP 3>= 3.0.5, PHP 4 >= 4.0.0)

Free all resources associated with a statement

int **OCIFreeStatement** (int stmt) \linebreak

OCIFreeStatement() returns **TRUE** if successful, or **FALSE** if unsuccessful.

OCIInternalDebug (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Enables or disables internal debug output

void **OCIInternalDebug** (int onoff) \linebreak

OCIInternalDebug() enables internal debug output. Set *onoff* to 0 to turn debug output off, 1 to turn it on.

OCILoadLob (PHP 4 >= 4.0.0)

Coming soon

string **OCILoadLob** (object lob) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCILogOff (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Disconnects from Oracle

int **OCILogOff** (int connection) \linebreak

OCILogOff() closes an Oracle connection.

OCILogon (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Establishes a connection to Oracle

int **OCILogon** (string username, string password [, string db]) \linebreak

OCILogon() returns a 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()**.

OCINewCollection (PHP 4 >= 4.0.6)

Coming soon

string **OCINewCollection** (int conn, string tdo [, string shema]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCINewCursor (PHP 3 >= 3.0.8, PHP 4 >= 4.0.0)

Return a new cursor (Statement-Handle)

int **OCINewCursor** (int conn) \linebreak

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);
}

OCIFreeStatement($stmt);
OCIFreeCursor($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>$deptno</TD>";
    ociexecute($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);
?>

```

OCINewDescriptor (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Initialize a new empty LOB or FILE descriptor

string **OCINewDescriptor** (int connection [, int type]) \linebreak

OCINewDescriptor() allocates storage to hold descriptors or LOB locators. Valid values for *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);

```

```

OCIExecute($stmt);
while ( OCIFetch($stmt) ) {
    $nrows = OCIRowCount($stmt);
    $delete = OCIParse($conn,"delete from $table where ROWID = :rid");
    OCIBindByName($delete,":rid",&$rowid,-1,OCI_B_ROWID);
    OCIExecute($delete);
    print "$nrows\n";
    if ( ($nrows % $commitsize) == 0 ) {
        OCICommit($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.php" method="post" enctype="multipart/form-data">
 * <input type="file" name="lob_upload">
 * ...
 */
if(!isset($lob_upload) || $lob_upload == 'none'){
?>
<form action="upload.php" 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

    // see also the features section on file upload,
    // if you would like to use secure uploads

    $conn = OCILogon($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, OCI_DEFAULT);
    if($lob->savefile($lob_upload)){
        OCICommit($conn);
        echo "Blob successfully uploaded\n";
    }else{
        echo "Couldn't upload Blob\n";
    }
    OCIFreeDesc($lob);

```

```

        OCIFreeStatement($stmt);
        OCILogoff($conn);
    }
?>

```

Example 2. OCINewDescriptor

```

<?php
/* Calling PL/SQL stored procedures which contain clobs as input
 * parameters (PHP 4 >= 4.0.6).
 * Example PL/SQL stored procedure signature is:
 *
 * PROCEDURE save_data
 *   Argument Name                Type                In/Out Default?
 *   -----
 *
 *   KEY                          NUMBER(38)        IN
 *   DATA                         CLOB              IN
 *
 */

$conn = OCILogon($user, $password);
$stmt = OCIParse($conn, "begin save_data(:key, :data); end;");
$clob = OCINewDescriptor($conn, OCI_D_LOB);
OCIBindByName($stmt, ':key', $key);
OCIBindByName($stmt, ':data', $clob, -1, OCI_B_CLOB);
$clob->WriteTemporary($data);
OCIExecute($stmt, OCI_DEFAULT);
OCICommit($conn);
$clob->close();
$clob->free();
OCIFreeStatement($stmt);
?>

```

OCINLogon (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Establishes a new connection to Oracle

int **OCINLogon** (string username, string password [, string db]) \linebreak

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()**.

OCINumCols (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Return the number of result columns in a statement

int **OCINumCols** (int stmt) \linebreak

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");
OCIExecute($stmt);
while ( OCIFetch($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";
?>
```

OCIParse (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Parse a query and return a statement

int **OCIParse** (int conn, string query) \linebreak

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 or PL/SQL block.

OCIPLogon (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Connect to an Oracle database using a persistent connection

int **OCIPLogon** (string username, string password [, string db]) \linebreak

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()**.

OCIResult (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Returns column value for fetched row

mixed **OCIResult** (int statement, mixed column) \linebreak

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).

OCIRollback (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Rolls back outstanding transactions

int **OCIRollback** (int connection) \linebreak

OCIRollback() rolls back all outstanding statements for Oracle connection *connection*.

OCIRowCount (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Gets the number of affected rows

int **OCIRowCount** (int statement) \linebreak

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");
OCIExecute($stmt);
print OCIRowCount($stmt) . " rows inserted.<BR>";
OCIFreeStatement($stmt);
$stmt = OCIParse($conn,"delete from emp2");
OCIExecute($stmt);
print OCIRowCount($stmt) . " rows deleted.<BR>";
```

```

OCICcommit($conn);
OCIFreeStatement($stmt);
$stmt = OCIParse($conn,"drop table emp2");
OCIExecute($stmt);
OCIFreeStatement($stmt);
OCILOGoff($conn);
print "</PRE></HTML>";
?>

```

OCISaveLob (PHP 4 >= 4.0.0)

Coming soon

string **OCISaveLob** (object lob) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCISaveLobFile (PHP 4 >= 4.0.0)

Coming soon

string **OCISaveLobFile** (object lob) \linebreak

Warning

This function is currently not documented; only the argument list is available.

OCIServerVersion (PHP 3>= 3.0.4, PHP 4 >= 4.0.0)

Return a string containing server version information

string **OCIServerVersion** (int conn) \linebreak

Example 1. OCI Server Version

```
<?php
    $conn = OCILogon("scott","tiger");
    print "Server Version: " . OCI Server Version($conn);
    OCI Log Off($conn);
?>
```

OCI Set Prefetch (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

Sets number of rows to be prefetched

```
int OCI Set Prefetch ( int stmt, int rows) \linebreak
```

Sets the number of top level rows to be prefetched. The default value is 1 row.

OCI Statement Type (PHP 3 >= 3.0.5, PHP 4 >= 4.0.0)

Return the type of an OCI statement

```
string OCI Statement Type ( int stmt) \linebreak
```

OCI Statement Type() 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. OCIStatementType() 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>";
?>
```

OCIWriteLobToFile (PHP 4 >= 4.0.0)

Coming soon

void **OCIWriteLobToFile** (object lob [, string filename [, int start [, int length]]]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

LXX. OpenSSL functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

Introduction

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. PHP-4.0.4pl1 requires OpenSSL $\geq 0.9.6$, but PHP-4.0.5 and greater with also work with OpenSSL $\geq 0.9.5$.

Note: Please keep in mind that this extension is still considered experimental!

OpenSSL offers many features that this module currently doesn't support. Some of these may be added in the future.

Key/Certificate parameters

Quite a few of the openssl functions require a key or a certificate parameter. PHP 4.0.5 and earlier have to use a key or certificate resource returned by one of the `openssl_get_XXX` functions. Later versions may use one of the following methods:

- Certificates
 1. An X.509 resource returned from `openssl_x509_read`
 2. A string having the format `file://path/to/cert.pem`; the named file must contain a PEM encoded certificate
 3. A string containing the content of a certificate, PEM encoded
- Public/Private Keys
 1. A key resource returned from `openssl_get_publickey()` or `openssl_get_privatekey()`
 2. For public keys only: an X.509 resource
 3. A string having the format `file://path/to/file.pem` - the named file must contain a PEM

encoded certificate/private key (it may contain both)

4. A string containing the content of a certificate/key, PEM encoded
5. For private keys, you may also use the syntax *array(\$key, \$passphrase)* where *\$key* represents a key specified using the *file://* or textual content notation above, and *\$passphrase* represents a string containing the passphrase for that private key

Certificate Verification

When calling a function that will verify a signature/certificate, the *cainfo* parameter is an array containing file and directory names that specify the locations of trusted CA files. If a directory is specified, then it must be a correctly formed hashed directory as the **openssl** command would use.

PKCS7 Flags/Constants

The S/MIME functions make use of flags which are specified using a bitfield which can include one or more of the following values:

Table 1. PKCS7 CONSTANTS

Constant	Description
PKCS7_TEXT	adds text/plain content type headers to encrypted/signed message. If decrypting or verifying, it strips those headers from the output - if the decrypted or verified message is not of MIME type text/plain then an error will occur.
PKCS7_BINARY	normally the input message is converted to "canonical" format which is effectively using CR and LF as end of line: as required by the S/MIME specification. When this option is present, no translation occurs. This is useful when handling binary data which may not be in MIME format.
PKCS7_NOINTERN	when verifying a message, certificates (if any) included in the message are normally searched for the signing certificate. With this option only the certificates specified in the <i>extracerts</i> parameter of <i>openssl_pkcs7_verify()</i> are used. The supplied certificates can still be used as untrusted CAs however.
PKCS7_NOVERIFY	do not verify the signers certificate of a signed message.

Constant	Description
PKCS7_NOCHAIN	do not chain verification of signers certificates: that is don't use the certificates in the signed message as untrusted CAs.
PKCS7_NOCERTS	when signing a message the signer's certificate is normally included - with this option it is excluded. This will reduce the size of the signed message but the verifier must have a copy of the signers certificate available locally (passed using the <i>extracerts</i> to <i>openssl_pkcs7_verify()</i> for example.
PKCS7_NOATTR	normally when a message is signed, a set of attributes are included which include the signing time and the supported symmetric algorithms. With this option they are not included.
PKCS7_DETACHED	When signing a message, use cleartext signing with the MIME type multipart/signed. This is the default if the <i>flags</i> parameter to <i>openssl_pkcs7_sign()</i> if you do not specify any flags. If you turn this option off, the message will be signed using opaque signing, which is more resistant to translation by mail relays but cannot be read by mail agents that do not support S/MIME.
PKCS7_NOSIGS	Don't try and verify the signatures on a message

Note: These constants were added in 4.0.6.

openssl_csr_export (PHP 4 >= 4.2.0)

Exports a CSR to file or a var

```
bool openssl_csr_export ( resource csr, string out [, bool notext]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_csr_export_to_file (PHP 4 >= 4.2.0)

Exports a CSR to file or a var

```
bool openssl_csr_export_to_file ( resource csr, string outfilename [, bool notext]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_csr_new (PHP 4 >= 4.2.0)

Generates a privkey and CSR

```
bool openssl_csr_new ( array dn, resource privkey [, array extraattribs [, array configargs]]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_csr_sign (PHP 4 >= 4.2.0)

Signs a cert with another CERT

resource **openssl_csr_sign** (mixed csr, mixed x509, mixed priv_key, long days) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_error_string (PHP 4 >= 4.0.6)

Return openssl error message

mixed **openssl_error_string** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns an error message string, or `FALSE` if there are no more error messages to return.

`openssl_error_string()` returns the last error from the openssl library. Error messages are stacked, so this function should be called multiple times to collect all of the information.

The parameters/return type of this function may change before it appears in a release version of PHP

Example 1. `openssl_error_string()` example

```
// lets assume you just called an openssl function that failed
while($msg = openssl_error_string())
    echo $msg . "<br />\n";
```

Note: This function was added in 4.0.6.

`openssl_free_key` (PHP 4 >= 4.0.4)

Free key resource

void **`openssl_free_key`** (resource *key_identifier*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`openssl_free_key()` frees the key associated with the specified *key_identifier* from memory.

`openssl_get_privatekey` (PHP 4 >= 4.0.4)

Prepare a PEM formatted private key for use

resource **`openssl_get_privatekey`** (mixed *key* [, string *passphrase*]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns a positive key resource 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 (PHP 4 >= 4.0.4)

Extract public key from certificate and prepare it for use

resource **openssl_get_publickey** (mixed certificate) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns a positive key resource 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 (PHP 4 >= 4.0.4)

Open sealed data

bool **openssl_open** (string sealed_data, string open_data, string env_key, mixed priv_key_id) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns `TRUE` on success, `FALSE` on failure. 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*, and fills *open_data* with the decrypted data. 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_pkcs7_decrypt` (PHP 4 >= 4.0.6)

Decrypts an S/MIME encrypted message

`bool openssl_pkcs7_decrypt (string infilename, string outfilename, mixed recipcert, mixed recipkey) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Decrypts the S/MIME encrypted message contained in the file specified by *infilename* using the certificate and it's associated private key specified by *recipcert* and *recipkey*.

The decrypted message is output to the file specified by *outfilename*

Example 1. openssl_pkcs7_decrypt() example

```
// $cert and $key are assumed to contain your personal certificate and private
// key pair, and that you are the recipient of an S/MIME message
$infilename = "encrypted.msg"; // this file holds your encrypted message
$outfilename = "decrypted.msg"; // make sure you can write to this file

if (openssl_pkcs7_decrypt($infilename, $outfilename, $cert, $key))
    echo "decrypted!";
else
    echo "failed to decrypt!";
```

Note: This function was added in 4.0.6.

openssl_pkcs7_encrypt (PHP 4 >= 4.0.6)

Encrypt an S/MIME message

bool **openssl_pkcs7_encrypt** (string infile, string outfile, mixed recipcerts, array headers [, long flags]) \line-break

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_pkcs7_encrypt() takes the contents of the file named *infile* and encrypts them using an RC2 40-bit cipher so that they can only be read by the intended recipients specified by *recipcerts*, which is either a lone X.509 certificate, or an array of X.509 certificates. *headers* is an array of headers that will be prepended to the data after it has been encrypted. *flags* can be used to specify options that affect the encoding process - see PKCS7 constants. *headers* can be either an associative array keyed by header name, or an indexed array, where each element contains a single header line.

Example 1. openssl_pkcs7_encrypt() example

```
// the message you want to encrypt and send to your secret agent
// in the field, known as nighthawk. You have his certificate
// in the file nighthawk.pem
$data = <<<EOD
```

Nighthawk,

Top secret, for your eyes only!

The enemy is closing in! Meet me at the cafe at 8.30am
to collect your forged passport!

HQ
EOD;

```
// load key
$key = implode("", file("nighthawk.pem"));

// save message to file
$fp = fopen("msg.txt", "w");
fwrite($fp, $data);
fclose($fp);

// encrypt it
if (openssl_pkcs7_encrypt("msg.txt", "enc.txt", $key,
    array("To" => "nighthawk@example.com", // keyed
    syntax
        "From: HQ <hq@example.com>", // indexed syntax
        "Subject" => "Eyes only")))
{
    // message encrypted - send it!
    exec(ini_get("sendmail_path") . " < enc.txt");
}
```

openssl_pkcs7_sign (PHP 4 >= 4.0.6)

sign an S/MIME message

bool **openssl_pkcs7_sign** (string *infile*, string *outfile*, mixed *signcert*, mixed *privkey*, array *headers* [, long *flags* [, string *extracertsfilename*]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_pkcs7_sign() takes the contents of the file named *infile* and signs them using the certificate and its matching private key specified by *signcert* and *privkey* parameters.

headers is an array of headers that will be prepended to the data after it has been signed (see `openssl_pkcs7_encrypt()` for more information about the format of this parameter.

flags can be used to alter the output - see PKCS7 constants - if not specified, it defaults to `PKCS7_DETACHED`.

extracerts specifies the name of a file containing a bunch of extra certificates to include in the signature which can for example be used to help the recipient to verify the certificate that you used.

Example 1. `openssl_pkcs7_sign()` example

```
// the message you want to sign so that recipient can be sure it was you that
// sent it
$data = <<<EOD
```

You have my authorization to spend \$10,000 on dinner expenses.

```
The CEO
EOD;
// save message to file
$fp = fopen("msg.txt", "w");
fwrite($fp, $data);
fclose($fp);
// encrypt it
if (openssl_pkcs7_sign("msg.txt", "signed.txt", "mycert.pem",
    array("mycert.pem", "mypassphrase"),
    array("To" => "joes@sales.com", // keyed syntax
        "From: HQ <ceo@sales.com>", // indexed syntax
        "Subject" => "Eyes only"))
{
    // message signed - send it!
    exec(ini_get("sendmail_path") . " < signed.txt");
}
```

Note: This function was added in 4.0.6.

`openssl_pkcs7_verify` (PHP 4 >= 4.0.6)

Verifies the signature of an S/MIME signed message

```
bool openssl_pkcs7_verify ( string filename, int flags [, string outfilename [, array cainfo [, string extracerts]]])
\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_pkcs7_verify() reads the S/MIME message contained in the filename specified by *filename* and examines the digital signature. It returns `TRUE` if the signature is verified, `FALSE` if it is not correct (the message has been tampered with, or the signing certificate is invalid), or `-1` on error.

flags can be used to affect how the signature is verified - see PKCS7 constants for more information.

If the *outfilename* is specified, it should be a string holding the name of a file into which the certificates of the persons that signed the messages will be stored in PEM format.

If the *cainfo* is specified, it should hold information about the trusted CA certificates to use in the verification process - see certificate verification for more information about this parameter.

If the *extracerts* is specified, it is the filename of a file containing a bunch of certificates to use as untrusted CAs.

Note: This function was added in 4.0.6.

openssl_pkey_export (PHP 4 >= 4.2.0)

Gets an exportable representation of a key into a string or file

`bool openssl_pkey_export (mixed key, mixed out [, string passphrase [, array config_args]]) \linebreak`

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_pkey_export_to_file (PHP 4 >= 4.2.0)

Gets an exportable representation of a key into a file

```
bool openssl_pkey_export_to_file ( mixed key, string outfilename [, string passphrase [, array config_args]])  
\linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_pkey_new (PHP 4 >= 4.2.0)

Generates a new private key

```
resource openssl_pkey_new ( [array configargs]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_private_decrypt (PHP 4 >= 4.0.6)

Decrypts data with private key

bool **openssl_private_decrypt** (string data, string crypted, mixed key [, int padding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_private_encrypt (PHP 4 >= 4.0.6)

Encrypts data with private key

bool **openssl_private_encrypt** (string data, string crypted, mixed key [, int padding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_public_decrypt (PHP 4 >= 4.0.6)

Decrypts data with public key

bool **openssl_public_decrypt** (string data, string crypted, resource key [, int padding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_public_encrypt (PHP 4 >= 4.0.6)

Encrypts data with public key

bool **openssl_public_encrypt** (string data, string crypted, mixed key [, int padding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_seal (PHP 4 >= 4.0.4)

Seal (encrypt) data

int **openssl_seal** (string data, string sealed_data, array env_keys, array pub_key_ids) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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 (PHP 4 >= 4.0.4)

Generate signature

```
bool openssl_sign ( string data, string signature, mixed priv_key_id) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns TRUE on success, 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 (PHP 4 >= 4.0.4)

Verify signature

int **openssl_verify** (string data, string signature, mixed pub_key_id) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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()`.

openssl_x509_check_private_key (PHP 4 >= 4.2.0)

Checks if a private key corresponds to a CERT

bool **openssl_x509_check_private_key** (mixed cert, mixed key) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_x509_checkpurpose (PHP 4 >= 4.0.6)

Verifies if a certificate can be used for a particular purpose

```
bool openssl_x509_checkpurpose ( mixed x509cert, int purpose, array cainfo [, string untrustedfile]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Returns `TRUE` if the certificate can be used for the intended purpose, `FALSE` if it cannot, or `-1` on error.

`openssl_x509_checkpurpose()` examines the certificate specified by `x509cert` to see if it can be used for the purpose specified by `purpose`.

`cainfo` should be an array of trusted CA files/dirs as described in Certificate Verification.

`untrustedfile`, if specified, is the name of a PEM encoded file holding certificates that can be used to help verify the certificate, although no trust is placed in the certificates that come from that file.

Table 1. openssl_x509_checkpurpose() purposes

Constant	Description
X509_PURPOSE_SSL_CLIENT	Can the certificate be used for the client side of an SSL connection?
X509_PURPOSE_SSL_SERVER	Can the certificate be used for the server side of an SSL connection?
X509_PURPOSE_NS_SSL_SERVER	Can the cert be used for Netscape SSL server?
X509_PURPOSE_SMIME_SIGN	Can the cert be used to sign S/MIME email?
X509_PURPOSE_SMIME_ENCRYPT	Can the cert be used to encrypt S/MIME email?
X509_PURPOSE_CRL_SIGN	Can the cert be used to sign a certificate revocation list (CRL)?
X509_PURPOSE_ANY	Can the cert be used for Any/All purposes?

These options are not bitfields - you may specify one only!

Note: This function was added in 4.0.6.

openssl_x509_export (PHP 4 >= 4.2.0)

Exports a CERT to file or a var

bool **openssl_x509_export** (mixed x509, string outfilename [, bool notext]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_x509_export_to_file (PHP 4 >= 4.2.0)

Exports a CERT to file or a var

bool **openssl_x509_export_to_file** (mixed x509, string outfilename [, bool notext]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

openssl_x509_free (PHP 4 >= 4.0.6)

Free certificate resource

```
void openssl_x509_free ( resource x509cert ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_x509_free() frees the certificate associated with the specified *x509cert* resource from memory.

Note: This function was added in 4.0.6.

openssl_x509_parse (PHP 4 >= 4.0.6)

Parse an X509 certificate and return the information as an array

```
array openssl_x509_parse ( mixed x509cert [, bool shortnames] ) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_x509_parse() returns information about the supplied *x509cert*, including fields such as subject name, issuer name, purposes, valid from and valid to dates etc. *shortnames* controls how the data is indexed in the array - if *shortnames* is `TRUE` (the default) then fields will be indexed with the short name form, otherwise, the long name form will be used - e.g.: CN is the shortname form of `commonName`.

The structure of the returned data is (deliberately) not yet documented, as it is still subject to change.

Note: This function was added in 4.0.6.

openssl_x509_read (PHP 4 >= 4.0.6)

Parse an X.509 certificate and return a resource identifier for it

resource **openssl_x509_read** (mixed x509certdata) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

openssl_x509_read() parses the certificate supplied by *x509certdata* and returns a resource identifier for it.

Note: This function was added in 4.0.6.

LXXI. Oracle functions

Ora_Bind (PHP 3, PHP 4 >= 4.0.0)

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]) \linebreak

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 >= 4.0.0)

close an Oracle cursor

int ora_close (int cursor) \linebreak

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 >= 4.0.0)

get name of Oracle result column

string **Ora_ColumnName** (int cursor, int column) \linebreak

Returns the name of the field/column *column* on the cursor *cursor*. The returned name is in all uppercase letters. Column 0 is the first column.

Ora_ColumnSize (PHP 3, PHP 4 >= 4.0.0)

get size of Oracle result column

int **Ora_ColumnSize** (int cursor, int column) \linebreak

Returns the size of the Oracle column *column* on the cursor *cursor*. Column 0 is the first column.

Ora_ColumnType (PHP 3, PHP 4 >= 4.0.0)

get type of Oracle result column

string **Ora_ColumnType** (int cursor, int column) \linebreak

Returns the Oracle data type name of the field/column *column* on the cursor *cursor*. Column 0 is the first column. 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 >= 4.0.0)

commit an Oracle transaction

int **ora_commit** (int conn) \linebreak

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 >= 4.0.0)

disable automatic commit

```
int ora_commitoff ( int conn) \linebreak
```

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_CommitOn (PHP 3, PHP 4 >= 4.0.0)

enable automatic commit

```
int ora_commiton ( int conn) \linebreak
```

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 >= 4.0.0)

Parse, Exec, Fetch

```
int ora_do ( int conn, string query) \linebreak
```

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 >= 4.0.0)

get Oracle error message

```
string Ora_Error ( int cursor_or_connection) \linebreak
```

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 >= 4.0.0)

get Oracle error code

```
int Ora_ErrorCode ( int cursor_or_connection) \linebreak
```

Returns the numeric error code of the last executed statement on the specified cursor or connection.

Note: Support for connection ids was added in 3.0.4.

Ora_Exec (PHP 3, PHP 4 >= 4.0.0)

execute parsed statement on an Oracle cursor

```
int ora_exec ( int cursor) \linebreak
```

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 >= 4.0.0)

fetch a row of data from a cursor

```
int ora_fetch ( int cursor) \linebreak
```

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 >= 4.0.0)

Fetch a row into the specified result array

`int ora_fetch_into (int cursor, array result [, int flags]) \linebreak`

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 >= 4.0.0)

get data from a fetched column

`mixed ora_getcolumn (int cursor, mixed column) \linebreak`

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 >= 4.0.0)

close an Oracle connection

`int ora_logoff (int connection) \linebreak`

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 >= 4.0.0)

open an Oracle connection

```
int ora_logon ( string user, string password) \linebreak
```

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<emphasis>@TNSNAME</emphasis>", "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_Numcols (PHP 3, PHP 4 >= 4.0.0)

Returns the number of columns

```
int ora_numcols ( int cursor_ind) \linebreak
```

`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 >= 4.0.0)

Returns the number of rows

```
int ora_numrows ( int cursor_ind) \linebreak
```

ora_numrows() returns the number of rows in a result.

Ora_Open (PHP 3, PHP 4 >= 4.0.0)

open an Oracle cursor

int ora_open (int connection) \linebreak

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 >= 4.0.0)

parse an SQL statement

int ora_parse (int cursor_ind, string sql_statement, int defer) \linebreak

This function parses an SQL statement or a PL/SQL block and associates it with the given cursor.

Returns `TRUE` on success, `FALSE` on failure.

See also `ora_exec()`, `ora_fetch()`, and `ora_do()`.

Ora_pLogon (PHP 3, PHP 4 >= 4.0.0)

Open a persistent Oracle connection

int ora_plogon (string user, string password) \linebreak

Establishes a persistent connection between PHP and an Oracle database with the given username and password.

See also `ora_logon()`.

Ora_Rollback (PHP 3, PHP 4 >= 4.0.0)

roll back transaction

int ora_rollback (int connection) \linebreak

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.

LXXII. 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_close (PHP 4 >= 4.0.3)

Closes the connection to ovrimos

```
void ovrimos_close ( int connection) \linebreak
```

ovrimos_close() is used to close the specified connection.

ovrimos_close() closes a connection to Ovrimos. This has the effect of rolling back uncommitted transactions.

ovrimos_commit (PHP 4 >= 4.0.3)

Commits the transaction

```
int ovrimos_commit ( int connection_id) \linebreak
```

ovrimos_commit() is used to commit the transaction.

ovrimos_commit() commits the transaction.

ovrimos_connect (PHP 4 >= 4.0.3)

Connect to the specified database

```
int ovrimos_connect ( string host, string db, string user, string password) \linebreak
```

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 Ovrimos 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_cursor (PHP 4 >= 4.0.3)

Returns the name of the cursor

int **ovrimos_cursor** (int result_id) \linebreak

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) \linebreak

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_execute (PHP 4 >= 4.0.3)

Executes a prepared SQL statement

bool **ovrimos_execute** (int result_id [, array parameters_array]) \linebreak

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_fetch_into (PHP 4 >= 4.0.3)

Fetches a row from the result set

bool **ovrimos_fetch_into** (int result_id, array result_array [, string how [, int rownumber]]) \linebreak

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

```
bool ovrimos_fetch_row ( int result_id [, int how [, int row_number]]) \linebreak
```

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_field_len (PHP 4 >= 4.0.3)

Returns the length of the output column

int **ovrimos_field_len** (int result_id, int field_number) \linebreak

ovrimos_field_len() is used to get the length of the output column with number *field_number*, in result *result_id*.

ovrimos_field_len() returns the length of the output column at the (1-based) index specified.

ovrimos_field_name (PHP 4 >= 4.0.3)

Returns the output column name

int **ovrimos_field_name** (int result_id, int field_number) \linebreak

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_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) \linebreak

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_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) \linebreak

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_free_result (PHP 4 >= 4.0.3)

Frees the specified result_id

bool **ovrimos_free_result** (int result_id) \linebreak

ovrimos_free_result() is used to free the result_id.

ovrimos_free_result() frees the specified result_id *result_id*. Returns TRUE.

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) \linebreak`

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. It currently sets this parameter the specified result set. Returns `TRUE`.

ovrimos_num_fields (PHP 4 >= 4.0.3)

Returns the number of columns

`int ovrimos_num_fields (int result_id) \linebreak`

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_num_rows (PHP 4 >= 4.0.3)

Returns the number of rows affected by update operations

`int ovrimos_num_rows (int result_id) \linebreak`

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_prepare (PHP 4 >= 4.0.3)

Prepares an SQL statement

`int ovrimos_prepare (int connection_id, string query) \linebreak`

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 Ovrimos 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 Ovrimos SQL Server, prepare a statement and the execute it.

ovrimos_result (PHP 4 >= 4.0.3)

Retrieves the output column

int **ovrimos_result** (int result_id, mixed field) \linebreak

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

bool **ovrimos_result_all** (int result_id [, string format]) \linebreak

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_rollback (PHP 4 >= 4.0.3)

Rolls back the transaction

```
int ovrimos_rollback ( int connection_id) \linebreak
```

ovrimos_rollback() is used to roll back the transaction.

ovrimos_rollback() rolls back the transaction.

LXXIII. 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 begun 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 ("cookienname", "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 >= 4.0.0)

Flush the output buffer

void **flush** (void) \linebreak

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.

Server modules for Apache like mod_gzip may do buffering of their own that will cause **flush()** to not result in data being sent immediately to the client.

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.

Some versions of Microsoft Internet Explorer will only start to display the page after they have received 256 bytes of output, so you may need to send extra whitespace before flushing to get those browsers to display the page.

ob_clean (PHP 4 >= 4.2.0)

Clean (erase) the output buffer

void **ob_clean** (void) \linebreak

This function discards the contents of the output buffer.

This function does not destroy the output buffer like `ob_end_clean()` does.

See also `ob_flush()`, `ob_end_flush()` and `ob_end_clean()`.

ob_end_clean (PHP 4 >= 4.0.0)

Clean (erase) the output buffer and turn off output buffering

void **ob_end_clean** (void) \linebreak

This function discards the contents of the output buffer and turns off output buffering.

See also `ob_start()`, `ob_clean()` and `ob_end_flush()`.

ob_end_flush (PHP 4 >= 4.0.0)

Flush (send) the output buffer and turn off output buffering

```
void ob_end_flush ( void) \linebreak
```

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()`, `ob_flush()` and `ob_end_clean()`.

ob_flush (PHP 4 >= 4.2.0)

Flush (send) the output buffer

```
void ob_flush ( void) \linebreak
```

This function will send the contents of the output buffer (if any). If you want to further process the buffer's contents you have to call `ob_get_contents()` before `ob_flush()` as the buffer contents are discarded after `ob_flush()` is called.

This function does not destroy the output buffer like `ob_end_flush()` does.

See also `ob_get_contents()`, `ob_clean()`, `ob_end_flush()` and `ob_end_clean()`.

ob_get_contents (PHP 4 >= 4.0.0)

Return the contents of the output buffer

```
string ob_get_contents ( void) \linebreak
```

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) \linebreak
```

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_get_level (PHP 4 >= 4.2.0)

Return the nesting level of the output buffering mechanism

```
int ob_get_level ( void) \linebreak
```

This will return the level of nested output buffering handlers.

See also ob_start() and ob_get_contents().

ob_gzhandler (PHP 4 >= 4.0.4)

ob_start callback function to gzip output buffer

```
string ob_gzhandler ( string buffer [, int mode]) \linebreak
```

Note: *mode* was added in PHP 4.0.5.

ob_gzhandler() is intended to be used as a callback function for ob_start() to help facilitate sending gz-encoded data to web browsers that support compressed web pages. Before **ob_gzhandler()** actually sends compressed data, it determines what type of content encoding the browser will accept ("gzip", "deflate" or none at all) and will return it's output accordingly. All browsers are supported since it's up to the browser to send the correct header saying that it accepts compressed web pages.

Example 1. ob_gzhandler() Example

```
<?php
ob_start("ob_gzhandler");

?>
<html>
<body>
<p>This should be a compressed page.
</html>
</body>
```

See also ob_start() and ob_end_flush().

ob_implicit_flush (PHP 4 >= 4.0.0)

Turn implicit flush on/off

```
void ob_implicit_flush ( [int flag] ) \linebreak
```

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()`.

ob_start (PHP 4 >= 4.0.0)

Turn on output buffering

```
void ob_start ( [string output_callback] ) \linebreak
```

This function will turn output buffering on. While output buffering is active no output is sent from the script (other than headers), 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 should return a string. The function will be called when `ob_end_flush()` is called, or when the output buffer is flushed to the browser at the end of the request. When *output_callback* is called, it will receive the contents of the output buffer as its parameter and is expected to return a new output buffer as a result, which will be sent to the browser.

Note: In PHP 4.0.4, `ob_gzhandler()` was introduced to facilitate sending gz-encoded data to web browsers that support compressed web pages. `ob_gzhandler()` determines what type of content encoding the browser will accept and will return its output accordingly.

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.

`ob_end_clean()`, `ob_end_flush()`, `ob_clean()`, `ob_flush()` and `ob_start()` may not be called from a callback function. If you call them from callback function, the behavior is undefined. If you would like to delete the contents of a buffer, return "" (a null string) from callback function.

Example 1. User defined callback function example

```
<?php

function callback($buffer) {

    // replace all the apples with oranges
    return (ereg_replace("apples", "oranges", $buffer));

}

ob_start("callback");

?>

<html>
<body>
<p>It's like comparing apples to oranges.
</body>
</html>

<?php

ob_end_flush();

?>
```

Would produce:

```
<html>
<body>
<p>It's like comparing oranges to oranges.
</body>
</html>
```

See also `ob_get_contents()`, `ob_end_flush()`, `ob_end_clean()`, `ob_implicit_flush()` and `ob_gzhandler()`.

LXXIV. Object property and method call overloading

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

The purpose of this extension is to allow overloading of object property access and method calls. Only one function is defined in this extension, `overload()` which takes the name of the class that should have this functionality enabled. The class named has to define appropriate methods if it wants to have this functionality: `__get()`, `__set()` and `__call()` respectively for getting/setting a property, or calling a method. This way overloading can be selective. Inside these handler functions the overloading is disabled so you can access object properties normally.

Some simple examples on using the `overload()` function:

Example 1. Overloading a PHP class

```
<?php

class OO
{
    var $a = 111;
    var $elem = array('b' => 9, 'c' => 42);

    // Callback method for getting a property
    function __get($prop_name, &$prop_value)
    {
        if (isset($this->elem[$prop_name])) {
            $prop_value = $this->elem[$prop_name];
            return true;
        } else {
            return false;
        }
    }

    // Callback method for setting a property
    function __set($prop_name, $prop_value)
    {
        $this->elem[$prop_name] = $prop_value;
        return true;
    }
}

// Here we overload the OO object
```

```
overload('OO');

$o = new OO;
print "\$o->a: $o->a\n"; // print: $o->a:
print "\$o->b: $o->b\n"; // print: $o->b: 9
print "\$o->c: $o->c\n"; // print: $o->c: 42
print "\$o->d: $o->d\n"; // print: $o->d:

// add a new item to the $elem array in OO
$o->x = 56;

// instantiate stdClass (it is built-in in PHP 4)
// $val is not overloaded!
$val = new stdClass;
$val->prop = 555;

// Set "a" to be an array with the $val object in it
// But __set() will put this in the $elem array
$o->a = array($val);
var_dump($o->a[0]->prop);

?>
```

Warning

As this is an experimental extension, not all things work. There is no `__call()` support currently, you can only overload the get and set operations for properties. You cannot invoke the original overloading handlers of the class, and `__set()` only works to one level of property access.

overload (PHP 4 >= 4.2.0)

Enable property and method call overloading for a class

void **overload** ([string *class_name*]) \linebreak

The **overload()** function will enable property and method call overloading for a class identified by *class_name*. See an example in the introductory section of this part.

LXXV. PDF functions

Introduction

The PDF functions in PHP can create PDF files using the PDFlib library created by Thomas Merz (<http://www.pdflib.com/corporate/tm.html>). PDFlib is available for download at <http://www.pdflib.com/pdflib/index.html>, but requires that you purchase a license for commercial use. The JPEG (<ftp://ftp.uu.net/graphics/jpeg/>) and TIFF (<http://www.libtiff.org/>) libraries are required to compile this extension. Please see the PDFlib installation section for more information about compiling PDF support into PHP.

The documentation in this section is only meant to be an overview of the available functions in the PDFlib library and should not be considered an exhaustive reference. Please consult the documentation included in the source distribution of PDFlib for the full and detailed explanation of each function here. It provides a very good overview of what PDFlib is capable of doing and contains the most up-to-date documentation of all functions.

All of the functions in PDFlib and the PHP module have identical function names and parameters. You will need to understand some of the basic concepts of PDF and PostScript to efficiently use this extension. 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. Please see the PDFlib documentation included with the source distribution of PDFlib for a more thorough explanation of the coordinate system used.

Please note that most of the PDF functions require a *pdf object* as it's first parameter. Please see the examples below for more information.

Note: An alternative PHP module for PDF document creation based on FastIO's (<http://www.fastio.com/>) ClibPDF is available. Please see the ClibPDF section for details. Note that ClibPDF has a slightly different API compared to PDFlib.

Confusion with old PDFlib versions

Starting with PHP 4.0.5, the PHP extension for PDFlib is officially supported by PDFlib GmbH. This means that all the functions described in the PDFlib manual (V3.00 or greater) are supported by PHP 4 with exactly the same meaning and the same parameters. Only the return values may differ from the PDFlib manual, because the PHP convention of returning `FALSE` was adopted. For compatibility reasons this binding for PDFlib still supports the old functions, but they should be replaced by their new versions. PDFlib GmbH will not support any problems arising from the use of these deprecated functions.

Table 1. Deprecated functions and its replacements

Old function	Replacement
--------------	-------------

Old function	Replacement
pdf_put_image()	Not needed anymore.
pdf_execute_image()	Not needed anymore.
pdf_get_annotation()	pdf_get_bookmark() using the same parameters.
pdf_get_font()	pdf_get_value() passing "font" as the second parameter.
pdf_get_fontsize()	pdf_get_value() passing "fontsize" as the second parameter.
pdf_get_fontname()	pdf_get_parameter() passing "fontname" as the second parameter.
pdf_set_info_creator()	pdf_set_info() passing "Creator" as the second parameter.
pdf_set_info_title()	pdf_set_info() passing "Title" as the second parameter.
pdf_set_info_subject()	pdf_set_info() passing "Subject" as the second parameter.
pdf_set_info_author()	pdf_set_info() passing "Author" as the second parameter.
pdf_set_info_keywords()	pdf_set_info() passing "Keywords" as the second parameter.
pdf_set_leading()	pdf_set_value() passing "leading" as the second parameter.
pdf_set_text_rendering()	pdf_set_value() passing "textrendering" as the second parameter.
pdf_set_text_rise()	pdf_set_value() passing "textrise" as the second parameter.
pdf_set_horiz_scaling()	pdf_set_value() passing "horizscaling" as the second parameter.
pdf_set_text_matrix()	Not available anymore
pdf_set_char_spacing()	pdf_set_value() passing "charspacing" as the second parameter.
pdf_set_word_spacing()	pdf_set_value() passing "wordspacing" as the second parameter.
pdf_set_transition()	pdf_set_parameter() passing "transition" as the second parameter.
pdf_open()	pdf_new() plus an subsequent call of pdf_open_file()
pdf_set_font()	pdf_findfont() plus an subsequent call of pdf_setfont()
pdf_set_duration()	pdf_set_value() passing "duration" as the second parameter.
pdf_open_gif()	pdf_open_image_file() passing "gif" as the second parameter.

Old function	Replacement
pdf_open_jpeg()	pdf_open_image_file() passing "jpeg" as the second parameter.
pdf_open_tiff()	pdf_open_image_file() passing "tiff" as the second parameter.
pdf_open_png()	pdf_open_image_file() passing "png" as the second parameter.
pdf_get_image_width()	pdf_get_value() passing "imagewidth" as the second parameter and the image as the third parameter.
pdf_get_image_height()	pdf_get_value() passing "imageheight" as the second parameter and the image as the third parameter.

PDFlib 3.x Installation Hints

When using version 3.x of PDFlib, you should configure PDFlib with the option `--enable-shared-pdf-lib`.

Issues with older versions of PDFlib

Any version of PHP 4 after March 9, 2000 does not support versions of PDFlib older than 3.0.

PDFlib 3.0 or greater is supported by PHP 3.0.19 and later.

Examples

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 to get started. It creates `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
$pdf = pdf_new();
pdf_open_file($pdf, "test.pdf");
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($pdf, 50, 740);
pdf_lineto($pdf, 330, 740);
pdf_stroke($pdf);
pdf_end_page($pdf);
pdf_close($pdf);
pdf_delete($pdf);
echo "<A HREF=getpdf.php>finished</A>";
?>

```

The script `getpdf.php` just returns the pdf document.

```

<?php
$len = filesize($filename);
header("Content-type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=foo.pdf");
readfile($filename);
?>

```

The PDFlib distribution contains a more complex example which creates a page with an analog clock. Here we use the in memory creation feature of PDFlib to alleviate the need to use temporary files. The example, converted to PHP from the PDFlib example, is as follows: (The same example is available in the CLibPDF documentation.)

Example 2. pdfclock example from PDFlib distribution

```

<?php
$radius = 200;
$margin = 20;
$pagecount = 10;

$pdf = pdf_new();

if (!pdf_open_file($pdf, "")) {
    print error;
    exit;
};

```



```

pdf_set_parameter($pdf, "warning", "true");

pdf_set_info($pdf, "Creator", "pdf_clock.php");
pdf_set_info($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, -(($time['seconds'] - 15.0) * 6.0));
pdf_moveto($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);

# to see some difference
sleep(1);
}

pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);

header("Content-type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=foo.pdf");
print $buf;

pdf_delete($pdf);
?>

```

pdf_add_annotation (PHP 3 >= 3.0.12, PHP 4 >= 4.0.0)

Deprecated: Adds annotation

pdf_add_outline() is replaced by pdf_add_note()

See also pdf_add_note().

pdf_add_bookmark (PHP 4)

Adds bookmark for current page

int **pdf_add_bookmark** (int pdf object, string text [, int parent [, int open]]) \linebreak

Add a nested bookmark under *parent*, or a new top-level bookmark if *parent* = 0. Returns a bookmark descriptor which may be used as parent for subsequent nested bookmarks. If *open* = 1, child bookmarks will be folded out, and invisible if *open* = 0.

pdf_add_launchlink (PHP 4 >= 4.0.5)

Add a launch annotation for current page

int **pdf_add_launchlink** (int pdf object, float llx, float lly, float urx, float ury, string filename) \linebreak

Add a launch annotation (to a target of arbitrary file type).

pdf_add_loclink (PHP 4 >= 4.0.5)

Add a link annotation for current page

int **pdf_add_loclink** (int pdf object, float llx, float lly, float urx, float ury, int page, string dest) \linebreak

Add a link annotation to a target within the current PDF file.

pdf_add_note (PHP 4 >= 4.0.5)

Add a note annotation for current page

int **pdf_add_note** (int pdf object, float llx, float lly, float urx, float ury, string contents, string title, string icon, int open) \linebreak

Add a note annotation. icon is one of of "comment", "insert", "note", "paragraph", "newparagraph", "key", or "help".

pdf_add_outline (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Adds bookmark for current page

Deprecated.

See pdf_add_bookmark().

pdf_add_pdflink (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Adds file link annotation for current page

```
int pdf_add_pdflink ( int pdf object, float llx, float lly, float urx, float ury, string filename, int page, string dest)
\linebreak
```

Add a file link annotation (to a PDF target).

pdf_add_thumbnail (PHP 4 >= 4.0.5)

Adds thumbnail for current page

```
int pdf_add_thumbnail ( int pdf object, int image) \linebreak
```

Add an existing image as thumbnail for the current page.

pdf_add_weblink (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Adds weblink for current page

```
int pdf_add_weblink ( int pdf object, float llx, float lly, float urx, float ury, string url) \linebreak
```

Add a weblink annotation to a target URL on the Web.

pdf_arc (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws an arc (counterclockwise)

void **pdf_arc** (resource pdf object, float x, float y, float r, float alpha, float beta) \linebreak
 Draw a counterclockwise circular arc from alpha to beta degrees
 See also: pdf_arcn()

pdf_arcn (PHP 4 >= 4.0.5)

Draws an arc (clockwise)

void **pdf_arc** (resource pdf object, float x, float y, float r, float alpha, float beta) \linebreak
 Draw a clockwise circular arc from alpha to beta degrees
 See also: pdf_arc()

pdf_attach_file (PHP 4 >= 4.0.5)

Adds a file attachment for current page

int **pdf_attach_file** (int pdf object, float llx, float lly, float urx, float ury, string filename, string description, string author, string mimetype, string icon) \linebreak

Add a file attachment annotation. icon is one of "graph", "paperclip", "pushpin", or "tag".

pdf_begin_page (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Starts new page

void **pdf_begin_page** (int pdf object, float width, float height) \linebreak

Add a new page to the document. The *width* and *height* are specified in points, which are 1/72 of an inch.

Table 1. Common Page Sizes in Points

name	size
A0	2380X3368
A1	1684X2380
A2	1190X1684
A3	842X1190
A4	595X842
A5	421X595

name	size
A6	297X421
B5	501X709
letter (8.5"X11")	612X792
legal (8.5"X14")	612X1008
ledger (17"X11")	1224X792
11"X17"	792X1224

pdf_begin_pattern (PHP 4 >= 4.0.5)

Starts new pattern

```
int pdf_begin_pattern ( int pdf object, float width, float height, float xstep, float ystep, int painttype) \linebreak
```

Starts a new pattern definition and returns a pattern handle. *width*, and *height* define the bounding box for the pattern. *xstep* and *ystep* give the repeated pattern offsets. *painttype*=1 means that the pattern has its own colour settings whereas a value of 2 indicates that the current colour is used when the pattern is applied.

pdf_begin_template (PHP 4 >= 4.0.5)

Starts new template

```
void pdf_begin_template ( int pdf object, float width, float height) \linebreak
```

Start a new template definition.

pdf_circle (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws a circle

```
void pdf_circle ( int pdf object, float x, float y, float r) \linebreak
```

Draw a circle with center (x, y) and radius r.

pdf_clip (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Clips to current path

void **pdf_clip** (int pdf object) \linebreak

Use the current path as clipping path.

pdf_close (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Closes a pdf object

void **pdf_close** (int pdf object) \linebreak

Close the generated PDF file, and free all document-related resources.

pdf_close_image (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Closes an image

void **pdf_close_image** (int pdf object, int image) \linebreak

Close an *image* retrieved with one of the **pdf_open_image***() functions.

pdf_close_pdi (PHP 4 >= 4.0.5)

Close the input PDF document

void **pdf_close_pdi** (int pdf object, int dochandle) \linebreak

Close all open page handles, and close the input PDF document.

pdf_close_pdi_page (PHP 4 >= 4.0.5)

Close the page handle

void **pdf_close_pdi_page** (int pdf object, int pagehandle) \linebreak

Close the page handle, and free all page-related resources.

pdf_closepath (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Closes path

void **pdf_closepath** (int pdf object) \linebreak

Close the current path.

pdf_closepath_fill_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Closes, fills and strokes current path

void **pdf_closepath_fill_stroke** (int pdf object) \linebreak

Close the path, fill, and stroke it.

pdf_closepath_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Closes path and draws line along path

void **pdf_closepath_stroke** (int pdf object) \linebreak

Close the path, and stroke it.

pdf_concat (PHP 4 >= 4.0.5)

Concatenate a matrix to the CTM

void **pdf_concat** (int pdf object, float a, float b, float c, float d, float e, float f) \linebreak

Concatenate a matrix to the CTM.

pdf_continue_text (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Outputs text in next line

void **pdf_continue_text** (int pdf object, string text) \linebreak

Print text at the next line. The spacing between lines is determined by the *leading* parameter.

pdf_curveto (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws a curve

void **pdf_curveto** (int pdf object, float x1, float y1, float x2, float y2, float x3, float y3) \linebreak

Draw a Bezier curve from the current point, using 3 more control points.

pdf_delete (PHP 4 >= 4.0.5)

Deletes a PDF object

void **pdf_delete** (int pdf object) \linebreak

Delete the PDF object, and free all internal resources.

pdf_end_page (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Ends a page

void **pdf_end_page** (int pdf object) \linebreak

Finish the page.

pdf_end_pattern (PHP 4 >= 4.0.5)

Finish pattern

void **pdf_end_pattern** (int pdf object) \linebreak

Finish the pattern definition.

pdf_end_template (PHP 4 >= 4.0.5)

Finish template

void **pdf_end_template** (int pdf object) \linebreak

Finish the template definition.

pdf_endpath (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Ends current path

Deprecated, use one of the stroke, fill, or clip functions instead.

pdf_fill (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills current path

```
void pdf_fill_stroke ( int pdf object) \linebreak
```

Fill the interior of the path with the current fill color.

pdf_fill_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills and strokes current path

```
void pdf_fill_stroke ( int pdf object) \linebreak
```

Fill and stroke the path with the current fill and stroke color.

pdf_findfont (PHP 4 >= 4.0.5)

Prepare font for later use with pdf_setfont().

```
int pdf_findfont ( int pdf object, string fontname, string encoding, int embed) \linebreak
```

Prepare a font for later use with pdf_setfont(). The metrics will be loaded, and if embed is nonzero, the font file will be checked, but not yet used. *encoding* is one of "buitin", "macroman", "winansi", "host", or a user-defined encoding name, or the name of a CMap.

pdf_findfont() returns a font handle or FALSE on error.

Example 1. pdf_findfont() example

```
<?php

$font = pdf_findfont($pdf, "Times New Roman", "winansi", 1);
if ($font) {
    pdf_setfont($pdf, $font, 10);
}

?>
```

pdf_get_buffer (PHP 4 >= 4.0.5)

Fetch the buffer containig the generated PDF data.

string **pdf_get_buffer** (int pdf object) \linebreak

Get the contents of the PDF output buffer. The result must be used by the client before calling any other PDFlib function.

pdf_get_font (PHP 4 >= 4.0.0)

Deprecated: font handling

Deprecated.

See pdf_get_value().

pdf_get_fontname (PHP 4 >= 4.0.0)

Deprecated: font handling

Deprecated.

See pdf_get_parameter().

pdf_get_fontsize (PHP 4 >= 4.0.0)

Deprecated: font handling

Deprecated.

See pdf_get_value().

pdf_get_image_height (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Returns height of an image

string **pdf_get_image_height** (int pdf object, int image) \linebreak

pdf_get_image_height() is deprecated, use pdf_get_value() instead.

pdf_get_image_width (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Returns width of an image

```
string pdf_get_image_width ( int pdf object, int image) \linebreak
```

The **pdf_get_image_width()** is deprecated, use **pdf_get_value()** instead.

pdf_get_majorversion (PHP 4 >= 4.2.0)

Returns the major version number of the PDFlib

```
int pdf_get_majorversion ( void) \linebreak
```

Returns the major version number of the PDFlib.

pdf_get_minorversion (PHP 4 >= 4.2.0)

Returns the minor version number of the PDFlib

```
int pdf_get_majorversion ( void) \linebreak
```

Returns the minor version number of the PDFlib.

pdf_get_parameter (PHP 4)

Gets certain parameters

```
string pdf_get_parameter ( int pdf object, string key [, float modifier]) \linebreak
```

Get the contents of some PDFlib parameter with string type.

pdf_get_pdi_parameter (PHP 4 >= 4.0.5)

Get some PDI string parameters

```
string pdf_get_pdi_parameter ( int pdf object, string key, int doc, int page, int index) \linebreak
```

Get the contents of some PDI document parameter with string type.

pdf_get_pdi_value (PHP 4 >= 4.0.5)

Gets some PDI numerical parameters

string **pdf_get_pdi_value** (int pdf object, string key, int doc, int page, int index) \linebreak

Get the contents of some PDI document parameter with numerical type.

pdf_get_value (PHP 4)

Gets certain numerical value

float **pdf_get_value** (int pdf object, string key [, float modifier]) \linebreak

Get the contents of some PDFlib parameter with float type.

pdf_initgraphics (PHP 4 >= 4.0.5)

Resets graphic state

void **pdf_initgraphics** (int pdf object) \linebreak

Reset all implicit color and graphics state parameters to their defaults.

pdf_lineto (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws a line

void **pdf_lineto** (int pdf object, float x, float y) \linebreak

Draw a line from the current point to (x, y).

pdf_makespotcolor (PHP 4 >= 4.0.5)

Makes a spotcolor

void **pdf_makespotcolor** (int pdf object, string spotname) \linebreak

Make a named spot color from the current color.

pdf_moveto (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets current point

```
void pdf_moveto ( int pdf object, float x, float y) \linebreak
```

Set the current point.

Note: The current point for graphics and the current text output position are maintained separately. See pdf_set_text_pos() to set the text output position.

pdf_new (PHP 4 >= 4.0.5)

Creates a new pdf object

```
int pdf_new ( ) \linebreak
```

Create a new PDF object, using default error handling and memory management.

pdf_open (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Open a new pdf object

pdf_open() is deprecated, use pdf_new() plus pdf_open_file() instead.

See also pdf_new(), pdf_open_file().

pdf_open_CCITT (PHP 4 >= 4.0.5)

Opens a new image file with raw CCITT data

```
int pdf_open_CCITT ( int pdf object, string filename, int width, int height, int BitReverse, int k, int Blacks1) \linebreak
```

Open a raw CCITT image.

pdf_open_file (PHP 4 >= 4.0.5)

Opens a new pdf object

int **pdf_open_file** (int pdf object [, string filename]) \linebreak

Create a new PDF file using the supplied file name. If *filename* is empty the PDF document will be generated in memory instead of on file. The result must be fetched by the client with the `pdf_get_buffer()` function.

The following example shows how to create a pdf document in memory and how to output it correctly.

Example 1. Creating a PDF document in memory

```
<?php

$pdf = pdf_new();

pdf_open_file($pdf);
pdf_begin_page($pdf, 595, 842);
pdf_set_font($pdf, "Times-Roman", 30, "host");
pdf_set_value($pdf, "textrendering", 1);
pdf_show_xy($pdf, "A PDF document created in memory!", 50, 750);
pdf_end_page($pdf);
pdf_close($pdf);

$data = pdf_get_buffer($pdf);

header("Content-type: application/pdf");
header("Content-disposition: inline; filename=test.pdf");
header("Content-length: " . strlen($data));

echo $data;

?>
```

pdf_open_gif (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Deprecated: Opens a GIF image

Deprecated.

See `pdf_open_image()`,

pdf_open_image (PHP 4 >= 4.0.5)

Versatile function for images

int pdf_open_image (int PDF-document, string imagetype, string source, string data, long length, int width, int height, int components, int bpc, string params) \linebreak

Use image data from a variety of data sources. Supported types are "jpeg", "ccitt", "raw". Supported sources are "memory", "fileref", "url". *len* is only used for type="raw", *params* is only used for type="ccitt".

pdf_open_image_file (PHP 3 CVS only, PHP 4 >= 4.0.0)

Reads an image from a file

int pdf_open_image_file (int PDF-document, string imagetype, string filename [, string stringparam [, string intparam]]) \linebreak

Open an image file. Supported types are "jpeg", "tiff", "gif", and "png". *stringparam* is either "", "mask", "masked", or "page". *intparam* is either 0, the image id of the applied mask, or the page.

pdf_open_jpeg (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Deprecated: Opens a JPEG image

Deprecated.

See also pdf_open_image(),

pdf_open_memory_image (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Opens an image created with PHP's image functions

int pdf_open_memory_image (int pdf object, int image) \linebreak

The **pdf_open_memory_image()** function takes an image created with the PHP's image functions and makes it available for the pdf object. 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_place_image()`.

pdf_open_pdi (PHP 4 >= 4.0.5)

Opens a PDF file

```
int pdf_open_pdi ( int pdf object, string filename, string stringparam, int intparam) \linebreak
```

Open an existing PDF document for later use.

pdf_open_pdi_page (PHP 4 >= 4.0.5)

Prepare a page

```
int pdf_open_pdi_page ( int pdf object, int dochandle, int pagenumber, string pagelabel) \linebreak
```

Prepare a page for later use with `pdf_place_image()`

pdf_open_png (PHP 4 >= 4.0.0)

Deprecated: Opens a PNG image

Deprecated.

See `pdf_open_image()`.

pdf_open_tiff (PHP 4 >= 4.0.0)

Deprecated: Opens a TIFF image

```
int pdf_open_tiff ( int PDF-document, string filename) \linebreak
```

Deprecated.

See also `pdf_open_image()`,

pdf_place_image (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Places an image on the page

```
void pdf_place_image ( int pdf object, int image, float x, float y, float scale) \linebreak
```

Place an image with the lower left corner at (x, y), and scale it.

pdf_place_pdi_page (PHP 4 >= 4.0.6)

Places an image on the page

```
void pdf_place_pdi_page ( int pdf object, int page, float x, float y, float sx, float sy) \linebreak
```

Place a PDF page with the lower left corner at (x, y), and scale it.

pdf_rect (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws a rectangle

```
void pdf_rect ( int pdf object, float x, float y, float width, float height) \linebreak
```

Draw a rectangle at lower left (x, y) with width and height.

pdf_restore (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Restores formerly saved environment

```
void pdf_restore ( int pdf object) \linebreak
```

Restore the most recently saved graphics state.

pdf_rotate (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets rotation

void **pdf_rotate** (int pdf object, float phi) \linebreak

Rotate the coordinate system by phi degrees.

pdf_save (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Saves the current environment

void **pdf_save** (int pdf object) \linebreak

Save the current graphics state.

pdf_scale (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets scaling

void **pdf_scale** (int pdf object, float x-scale, float y-scale) \linebreak

Scale the coordinate system.

pdf_set_border_color (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Sets color of border around links and annotations

void **pdf_set_border_color** (int pdf object, float red, float green, float blue) \linebreak

Set the border color for all kinds of annotations.

pdf_set_border_dash (PHP 4)

Sets dash style of border around links and annotations

void **pdf_set_border_dash** (int pdf object, float black, float white) \linebreak

Set the border dash style for all kinds of annotations. See pdf_setdash().

pdf_set_border_style (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Sets style of border around links and annotations

void **pdf_set_border_style** (int pdf object, string style, float width) \linebreak

Set the border style for all kinds of annotations. *style* is "solid" or "dashed".

pdf_set_char_spacing (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Sets character spacing

Deprecated.

See also pdf_set_value(),

pdf_set_duration (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Sets duration between pages

Deprecated.

See pdf_set_value().

pdf_set_font (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Selects a font face and size

Deprecated. You should use pdf_findfont() plus pdf_setfont() instead.

See pdf_findfont(), pdf_setfont().

pdf_set_horiz_scaling (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets horizontal scaling of text

void **pdf_set_horiz_scaling** (int pdf object, float scale) \linebreak

Deprecated.

See also pdf_set_value(),

pdf_set_info (PHP 4)

Fills a field of the document information

```
void pdf_set_info ( int pdf object, string key, string value) \linebreak
```

Fill document information field key with value. *key* is one of "Subject", "Title", "Creator", "Author", "Keywords", or a user-defined key.

pdf_set_info_author (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills the author field of the document

```
bool pdf_set_info_author ( int pdfdoc, string author) \linebreak
```

This function is deprecate, use pdf_set_info() instead.

pdf_set_info_creator (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills the creator field of the document

```
bool pdf_set_info_creator ( int pdfdoc, string creator) \linebreak
```

This function is deprecate, use pdf_set_info() instead.

pdf_set_info_keywords (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills the keywords field of the document

```
bool pdf_set_info_keywords ( int pdfdoc, string keywords) \linebreak
```

This function is deprecate, use pdf_set_info() instead.

pdf_set_info_subject (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills the subject field of the document

```
bool pdf_set_info_subject ( int pdfdoc, string subject) \linebreak
```

This function is deprecate, use pdf_set_info() instead.

pdf_set_info_title (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Fills the title field of the document

```
bool pdf_set_info_title ( int pdfdoc, string title) \linebreak
```

This function is deprecate, use pdf_set_info() instead.

pdf_set_leading (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Sets distance between text lines

Deprecated.

See also pdf_set_value(),

pdf_set_parameter (PHP 4 >= 4.0.0)

Sets certain parameters

```
void pdf_set_parameter ( int pdf object, string key, string value) \linebreak
```

Set some PDFlib parameter with string type.

pdf_set_text_pos (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets text position

```
void pdf_set_text_pos ( int pdf object, float x, float y) \linebreak
```

Set the text output position.

pdf_set_text_rendering (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Determines how text is rendered

Deprecated.

See pdf_set_value(),

pdf_set_text_rise (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Deprecated: Sets the text rise

Deprecated.

See pdf_set_value(),

pdf_set_text_matrix (PHP 3>= 3.0.6)

Deprecated: Sets the text matrix

See pdf_set_paramter().

pdf_set_value (PHP 4)

Sets certain numerical value

void **pdf_set_value** (int pdf object, string key, float value) \linebreak

Set the value of some PDFlib parameter with float type.

pdf_set_word_spacing (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Depreciated: Sets spacing between words

Deprecated.

See also pdf_set_value(),

pdf_setcolor (PHP 4 >= 4.0.5)

Sets fill and stroke color

void **pdf_setcolor** (int pdf object, string type, string colorspace, float c1 [, float c2 [, float c3 [, float c4]])
\linebreak

Set the current color space and color. The parameter *type* can be "fill", "stroke", or "both" to specify that the color is set for filling, stroking or both filling and stroking. The parameter *colorspace* can be gray, rgb, cmyk, spot or pattern. The parameters *c1*, *c2*, *c3* and *c4* represent the color

components for the color space specified by *colorspace*. Except as otherwise noted, the color components are floating-point values that range from 0 to 1.

For *gray* only *c1* is used.

For *rgb* parameters *c1*, *c2*, and *c3* specify the red, green and blue values respectively.

```
// Set fill and stroke colors to white.
pdf_setcolor($pdf, "both", "rgb", 1, 1, 1);
```

For *cmymk*, parameters *c1*, *c2*, *c3*, and *c4* are the cyan, magenta, yellow and black values, respectively.

```
// Set fill and stroke colors to white.
pdf_setcolor($pdf, "both", "cmymk", 0, 0, 0, 1);
```

For *spot*, *c1* should be a spot color handles returned by `pdf_makespotcolor()` and *c2* is a tint value between 0 and 1.

For *pattern*, *c1* should be a pattern handle returned by `pdf_begin_pattern()`.

pdf_setdash (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets dash pattern

```
void pdf_setdash ( int pdf object, float b, float w) \linebreak
```

Set the current dash pattern to *b* black and *w* white units.

pdf_setflat (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets flatness

```
void pdf_setflat ( int pdf object, float flatness) \linebreak
```

Set the flatness to a value between 0 and 100 inclusive.

pdf_setfont (PHP 4 >= 4.0.5)

Set the current font

```
void pdf_setfont ( int pdf object, int font, float size) \linebreak
```

Set the current font in the given size, using a *font* handle returned by pdf_findfont()

See Also: pdf_findfont().

pdf_setgray (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets drawing and filling color to gray value

```
void pdf_setgray ( int pdf object, float gray) \linebreak
```

Set the current fill and stroke color.

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_setgray_fill (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets filling color to gray value

```
void pdf_setgray_fill ( int pdf object, float gray) \linebreak
```

Set the current fill color to a gray value between 0 and 1 inclusive.

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_setgray_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets drawing color to gray value

```
void pdf_setgray_stroke ( int pdf object, float gray) \linebreak
```

Set the current stroke color to a gray value between 0 and 1 inclusive

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_setlinecap (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets linecap parameter

```
void pdf_setlinecap ( int pdf object, int linecap) \linebreak
```

Set the *linecap* parameter to a value between 0 and 2 inclusive.

pdf_setlinejoin (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets linejoin parameter

```
void pdf_setlinejoin ( int pdf object, long linejoin) \linebreak
```

Set the line join parameter to a value between 0 and 2 inclusive.

pdf_setlinewidth (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets line width

```
void pdf_setlinewidth ( int pdf object, float width) \linebreak
```

Set the current linewidth to width.

pdf_setmatrix (PHP 4 >= 4.0.5)

Sets current transformation matrix

```
void pdf_setmatrix ( int pdf object, float a, float b, float c, float d, float e, float f) \linebreak
```

Explicitly set the current transformation matrix.

pdf_setmiterlimit (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets miter limit

```
void pdf_setmiterlimit ( int pdf object, float miter) \linebreak
```

Set the miter limit to a value greater than or equal to 1.

pdf_setpolydash (PHP 4 >= 4.0.5)

Sets complicated dash pattern

```
void pdf_setpolydash ( int pdf object, float * dasharray) \linebreak
```

Set a more complicated dash pattern defined by an array.

pdf_setrgbcolor (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets drawing and filling color to rgb color value

```
void pdf_setrgbcolor ( int pdf object, float red value, float green value, float blue value) \linebreak
```

Set the current fill and stroke color to the supplied RGB values.

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_setrgbcolor_fill (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets filling color to rgb color value

```
void pdf_setrgbcolor_fill ( int pdf object, float red value, float green value, float blue value) \linebreak
```

Set the current fill color to the supplied RGB values.

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_setrgbcolor_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets drawing color to rgb color value

```
void pdf_setrgbcolor_stroke ( int pdf object, float red value, float green value, float blue value) \linebreak
```

Set the current stroke color to the supplied RGB values.

Note: PDFlib V4.0: Deprecated, use pdf_setcolor() instead.

pdf_show (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Output text at current position

```
void pdf_show ( int pdf object, string text) \linebreak
```

Print text in the current font and size at the current position.

pdf_show_boxed (PHP 4 >= 4.0.0)

Output text in a box

```
int pdf_show_boxed ( int pdf object, string text, float left, float top, float width, float height, string hmode [, string feature]) \linebreak
```

Format text in the current font and size into the supplied text box according to the requested formatting mode, which must be one of "left", "right", "center", "justify", or "fulljustify". If width and height are 0, only a single line is placed at the point (left, top) in the requested mode.

Returns the number of characters that did not fit in the specified box. Returns 0 if all characters fit or the *width* and *height* parameters were set to 0 for single-line formatting.

pdf_show_xy (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Output text at given position

```
void pdf_show_xy ( int pdf object, string text, float x, float y) \linebreak
```

Print text in the current font at (x, y).

pdf_skew (PHP 4 >= 4.0.0)

Skews the coordinate system

```
void pdf_skew ( int pdf object, float alpha, float beta) \linebreak
```

Skew the coordinate system in x and y direction by alpha and beta degrees.

pdf_stringwidth (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Returns width of text using current font

float **pdf_stringwidth** (int pdf object, string text [, int font [, float size]]) \linebreak

Returns the width of *text* using the last font set by `pdf_setfont()`. If the optional parameters *font* and *size* are specified, the width will be calculated using that font and size instead. Please note that *font* is a font handle returned by `pdf_findfont()`.

Note: Both the *font* and *size* parameters must used together.

See Also: `pdf_setfont()` and `pdf_findfont()`.

pdf_stroke (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Draws line along path

void **pdf_stroke** (int pdf object) \linebreak

Stroke the path with the current color and line width, and clear it.

pdf_translate (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Sets origin of coordinate system

void **pdf_translate** (int pdf object, float tx, float ty) \linebreak

Translate the origin of the coordinate system.

LXXVI. 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.

pfpro_cleanup (PHP 4 >= 4.0.2)

Shuts down the Payflow Pro library

void **pfpro_cleanup** (void) \linebreak

pfpro_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 **pfpro_cleanup()** after your script terminates.

See also `pfpro_init()`.

pfpro_init (PHP 4 >= 4.0.2)

Initialises the Payflow Pro library

void **pfpro_init** (void) \linebreak

pfpro_init() is used to initialise the Payflow Pro library. You may omit this call, in which case this extension will automatically call **pfpro_init()** before the first transaction.

See also `pfpro_cleanup()`.

pfpro_process (PHP 4 >= 4.0.2)

Process a transaction with Payflow Pro

array **pfpro_process** (array parameters [, string address [, int port [, int timeout [, string proxy address [, int proxy port [, string proxy logon [, string proxy password]]]]]]) \linebreak

Returns: An associative array containing the response

pfpro_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

pfpro_init();

$transaction = array(USER => 'mylogin',
    PWD => 'mypassword',
    TRXTYPE => 'S',
    TENDER => 'C',
    AMT => 1.50,
    ACCT => '4111111111111111',
    EXPDATE => '0904'
);

$response = pfpro_process($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);

pfpro_cleanup();

?>
```

pfpro_process_raw (PHP 4 >= 4.0.2)

Process a raw transaction with Payflow Pro

string **pfpro_process_raw** (string parameters [, string address [, int port [, int timeout [, string proxy address [, int proxy port [, string proxy logon [, string proxy password]]]]]]]) \linebreak

Returns: A string containing the response.

pfpro_process_raw() processes a raw transaction string with Payflow Pro. You should really use **pfpro_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 **pfpro_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 **pfpro_process()** instead.

Example 1. Payflow Pro raw example

```
<?php

pfpro_init();

$response = pfpro_process("USER=mylogin&PWD[5]=m&ndy&TRXTYPE=S&TENDER=C&AMT=1.50&ACCT=411111

if (!$response) {
    die("Couldn't establish link to Verisign.\n");
}

echo "Verisign raw response was ".$response;

pfpro_cleanup();

?>
```

pfpro_version (PHP 4 >= 4.0.2)

Returns the version of the Payflow Pro software

string **pfpro_version** (void) \linebreak

pfpro_version() returns the version string of the Payflow Pro library. At the time of writing, this was L211.

LXXVII. PHP Options&Information

assert (PHP 4 >= 4.0.0)

Checks if assertion is FALSE

```
int assert ( string|bool assertion) \linebreak
```

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.

Assertions 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 correctly 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.

The `assert_options()` function and/or `ASSERT_CALLBACK` configuration directive allow a callback function to be set to handle failed assertions.

assert() callbacks are particularly useful for building automated test suites because they allow you to easily capture the code passed to the assertion, along with information on where the assertion was made. While this information can be captured via other methods, using assertions makes it much faster and easier!

The callback function should accept three arguments. The first argument will contain the file the assertion failed in. The second argument will contain the line the assertion failed on and the third argument will contain the expression that failed (if any - literal values such as 1 or "two" will not be passed via this argument)

Example 1. Handle a failed assertion with a custom handler

```
<?php
// Active assert and make it quiet
assert_options (ASSERT_ACTIVE, 1);
assert_options (ASSERT_WARNING, 0);
assert_options (ASSERT_QUIET_EVAL, 1);

// Create a handler function
function my_assert_handler ($file, $line, $code) {
    echo "<hr>Assertion Failed:
        File '$file'<br>
        Line '$line'<br>
        Code '$code'<br><hr>";
}

// Set up the callback
assert_options (ASSERT_CALLBACK, 'my_assert_handler');
```

```
// Make an assertion that should fail
assert ('mysql_query ("");');
?>
```

assert_options (PHP 4 >= 4.0.0)

Set/get the various assert flags

mixed **assert_options** (int what [, mixed value]) \linebreak

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.

dl (PHP 3, PHP 4 >= 4.0.0)

Loads a PHP extension at runtime

int **dl** (string library) \linebreak

Loads the PHP extension defined in *library*. See also the Extension Loading Directives

extension_loaded (PHP 3 >= 3.0.10, PHP 4 >= 4.0.0)

Find out whether an extension is loaded

bool **extension_loaded** (string *name*) \linebreak

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.

get_cfg_var (PHP 3, PHP 4 >= 4.0.0)

Gets the value of a PHP configuration option

string **get_cfg_var** (string *varname*) \linebreak

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.

See also `ini_get()`.

get_current_user (PHP 3, PHP 4 >= 4.0.0)

Gets the name of the owner of the current PHP script

string **get_current_user** (void) \linebreak

Returns the name of the owner of the current PHP script.

See also `getmyuid()`, `getmygid()`, `getmypid()`, `getmyinode()`, and `getlastmod()`.

get_defined_constants (PHP 4 >= 4.1.0)

Returns an associative array with the names of all the constants and their values

array **get_defined_constants** (void) \linebreak

This function returns the names and values of all the constants currently defined. This includes those created by extensions as well as those created with the `define()` function.

For example the line below

```
print_r (get_defined_constants());
```

will print a list like:

```
Array
(
    [E_ERROR] => 1
    [E_WARNING] => 2
    [E_PARSE] => 4
    [E_NOTICE] => 8
    [E_CORE_ERROR] => 16
    [E_CORE_WARNING] => 32
    [E_COMPILE_ERROR] => 64
    [E_COMPILE_WARNING] => 128
    [E_USER_ERROR] => 256
    [E_USER_WARNING] => 512
    [E_USER_NOTICE] => 1024
    [E_ALL] => 2047
    [TRUE] => 1
)
```

See also `get_loaded_extensions()`, `get_defined_functions()` and `get_defined_vars()`.

get_extension_funcs (PHP 4 >= 4.0.0)

Returns an array with the names of the functions of a module

array **get_extension_funcs** (string *module_name*) \linebreak

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_included_files (PHP 4 >= 4.0.0)

Returns an array with the names of included or required files

array **get_included_files** (void) \linebreak

Returns an array of the names of all files that have been included using `include()`, `include_once()`, `require()` or `require_once()`.

Files that are included or required multiple times only show up once in the returned array.

Note: Files included using the `auto_prepend_file` configuration directive are not included in the returned array.

Example 1. get_included_files() Example

```
<?php

include("test1.php");
include_once("test2.php");
require("test3.php");
require_once("test4.php");

$included_files = get_included_files();

foreach($included_files as $filename) {
    echo "$filename\n";
}

?>
```

will generate the following output:

```
test1.php
test2.php
test3.php
```

test4.php

Note: In PHP 4.0.1pl2 and previous versions **get_included_files()** assumed that the required files ended in the extension `.php`; other extensions would not be returned. The array returned by **get_included_files()** was an associative array and only listed files included by `include()` and `include_once()`.

See also: `include()`, `include_once()`, `require()`, `require_once()`, and `get_required_files()`.

get_loaded_extensions (PHP 4 >= 4.0.0)

Returns an array with the names of all modules compiled and loaded

array **get_loaded_extensions** (void) \linebreak

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_magic_quotes_gpc (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Gets the current active configuration setting of magic quotes gpc

long **get_magic_quotes_gpc** (void) \linebreak

Returns the current active configuration setting of `magic_quotes_gpc` (0 for off, 1 for on).

See also `get_magic_quotes_runtime()` and `set_magic_quotes_runtime()`.

get_magic_quotes_runtime (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Gets the current active configuration setting of magic_quotes_runtime

long **get_magic_quotes_runtime** (void) \linebreak

Returns the current active configuration setting of `magic_quotes_runtime` (0 for off, 1 for on).

See also `get_magic_quotes_gpc()` and `set_magic_quotes_runtime()`.

get_required_files (PHP 4 >= 4.0.0)

Returns an array with the names of included or required files

array **get_required_files** (void) \linebreak

As of PHP 4.0.4, this function is an alias for `get_included_files()`

In PHP 4.0.1pl2 and previous versions **get_required_files()** assumed that the required files ended in the extension `.php`, other extensions would not be returned. The array returned by **get_required_files()** was an associative array and only listed files included by `require()` and `require_once()`.

See also: `require()`, `require_once()`, `include()`, `include_once()`, and `get_included_files()`.

getenv (PHP 3, PHP 4 >= 4.0.0)

Gets the value of an environment variable

string **getenv** (string varname) \linebreak

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.

See also `putenv()`.

getlastmod (PHP 3, PHP 4 >= 4.0.0)

Gets time of last page modification

```
int getlastmod ( void) \linebreak
```

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()`, `getmygid()`, `get_current_user()`, `getmyinode()`, and `getmypid()`.

getmygid (PHP 4 >= 4.1.0)

Get PHP script owner's GID

```
int getmygid ( void) \linebreak
```

Returns the group ID of the current script, or `FALSE` on error.

See also `getmyuid()`, `getmypid()`, `get_current_user()`, `getmyinode()`, and `getlastmod()`.

getmyinode (PHP 3, PHP 4 >= 4.0.0)

Gets the inode of the current script

```
int getmyinode ( void) \linebreak
```

Returns the current script's inode, or `FALSE` on error.

See also `getmygid()`, `getmyuid()`, `get_current_user()`, `getmypid()`, and `getlastmod()`.

Note: This function is not implemented on Windows platforms.

getmypid (PHP 3, PHP 4 >= 4.0.0)

Gets PHP's process ID

```
int getmypid ( void) \linebreak
```

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 `getmygid()`, `getmyuid()`, `get_current_user()`, `getmyinode()`, and `getlastmod()`.

getmyuid (PHP 3, PHP 4 >= 4.0.0)

Gets PHP script owner's UID

```
int getmyuid ( void) \linebreak
```

Returns the user ID of the current script, or `FALSE` on error.

See also `getmygid()`, `getmypid()`, `get_current_user()`, `getmyinode()`, and `getlastmod()`.

getrusage (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Gets the current resource usages

```
array getrusage ( [int who]) \linebreak
```

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 >= 4.0.0)

Changes the value of a configuration option

string **ini_alter** (string varname, string newvalue) \linebreak

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_get_all()`, `ini_restore()`, and `ini_set()`

ini_get (PHP 4 >= 4.0.0)

Gets the value of a configuration option

string **ini_get** (string varname) \linebreak

Returns the value of the configuration option on success, an empty string on failure.

See also `get_cfg_var()`, `ini_get_all()`, `ini_alter()`, `ini_restore()`, and `ini_set()`

ini_get_all (PHP 4 >= 4.2.0)

Gets all configuration options

array **ini_get_all** ([string extension]) \linebreak

Returns all the registered configuration options as an associative array. If optional *extension* parameter is set, returns only options specific for that extension.

See also: `ini_alter()`, `ini_restore()`, `ini_get()`, and `ini_set()`

ini_restore (PHP 4 >= 4.0.0)

Restores the value of a configuration option

string **ini_restore** (string varname) \linebreak

Restores a given configuration option to its original value.

See also: `ini_alter()`, `ini_get()`, `ini_get_all()`, and `ini_set()`

ini_set (PHP 4 >= 4.0.0)

Sets the value of a configuration option

string **ini_set** (string varname, string newvalue) \linebreak

Sets the value of the given configuration option. Returns the old value on success, `FALSE` on failure. The configuration option will keep this new value during the script's execution, and will be restored at the script's ending.

Not all the available options can be changed using `ini_set()`. Below is a table with a list of all PHP options (as of PHP 4.0.5), indicating which ones can be changed/set and at what level.

Table 1. Configuration options

Name	Default	Changeable
foo	"1"	PHP_INI_ALL
bar	"bah"	PHP_INI_SYSTEM
Always declare those message handlers using <code>PHP_INI_MH()</code>	astheymight	
message handler - that's the fifth argument to <code>PHP_INI_ENTRY()</code> . It is		
com.allow_dcom	"0"	PHP_INI_SYSTEM
com.autoregister_typelib	"0"	PHP_INI_SYSTEM
com.autoregister_verbose	"0"	PHP_INI_SYSTEM
com.autoregister_casesensitive	"1"	PHP_INI_SYSTEM
com.typelib_file	""	PHP_INI_SYSTEM

Name	Default	Changeable
crack.default_dictionary	NULL	PHP_INI_SYSTEM
exif.encode_unicode	"ISO-8859-15"	PHP_INI_ALL
exif.decode_unicode_motorola	"UCS-2BE"	PHP_INI_ALL
exif.decode_unicode_intel	"UCS-2LE"	PHP_INI_ALL
exif.encode_jis	""	PHP_INI_ALL
exif.decode_jis_motorola	"JIS"	PHP_INI_ALL
exif.decode_jis_intel	"JIS"	PHP_INI_ALL
fbsql.allow_persistent	"1"	PHP_INI_SYSTEM
fbsql.generate_warnings	"0"	PHP_INI_SYSTEM
fbsql.autocommit	"1"	PHP_INI_SYSTEM
fbsql.max_persistent	"-1"	PHP_INI_SYSTEM
fbsql.max_links	"128"	PHP_INI_SYSTEM
fbsql.max_connections	"128"	PHP_INI_SYSTEM
fbsql.max_results	"128"	PHP_INI_SYSTEM
fbsql.batchSize	"1000"	PHP_INI_SYSTEM
fbsql.default_host	NULL	PHP_INI_SYSTEM
fbsql.default_user	"_SYSTEM"	PHP_INI_SYSTEM
fbsql.default_password	""	PHP_INI_SYSTEM
fbsql.default_database	""	PHP_INI_SYSTEM
fbsql.default_database_password	""	PHP_INI_SYSTEM
hwapi.allow_persistent	"0"	PHP_INI_SYSTEM
hyerwave.allow_persistent	"0"	PHP_INI_SYSTEM
hyperwave.default_port	"418"	PHP_INI_ALL
iconv.input_encoding	ICONV_INPUT_ENCODING	PHP_INI_ALL
iconv.output_encoding	ICONV_OUTPUT_ENCODING	PHP_INI_ALL
iconv.internal_encoding	ICONV_INTERNAL_ENCODING	PHP_INI_ALL
ifx.allow_persistent	"1"	PHP_INI_SYSTEM
ifx.max_persistent	"-1"	PHP_INI_SYSTEM
ifx.max_links	"-1"	PHP_INI_SYSTEM
ifx.default_host	NULL	PHP_INI_SYSTEM
ifx.default_user	NULL	PHP_INI_SYSTEM
ifx.default_password	NULL	PHP_INI_SYSTEM
ifx.blobinfile	"1"	PHP_INI_ALL
ifx.textasvarchar	"0"	PHP_INI_ALL
ifx.byteasvarchar	"0"	PHP_INI_ALL
ifx.charasvarchar	"0"	PHP_INI_ALL
ifx.nullformat	"0"	PHP_INI_ALL
ingres.allow_persistent	"1"	PHP_INI_SYSTEM

Name	Default	Changeable
ingres.max_persistent	"-1"	PHP_INI_SYSTEM
ingres.max_links	"-1"	PHP_INI_SYSTEM
ingres.default_database	NULL	PHP_INI_ALL
ingres.default_user	NULL	PHP_INI_ALL
ingres.default_password	NULL	PHP_INI_ALL
ibase.allow_persistent	"1"	PHP_INI_SYSTEM
ibase.max_persistent	"-1"	PHP_INI_SYSTEM
ibase.max_links	"-1"	PHP_INI_SYSTEM
ibase.default_user	NULL	PHP_INI_ALL
ibase.default_password	NULL	PHP_INI_ALL
ibase.timestampformat	"%m/%d/%Y%H:%M:%S"	PHP_INI_ALL
ibase.dateformat	"%m/%d/%Y"	PHP_INI_ALL
ibase.timeformat	"%H:%M:%S"	PHP_INI_ALL
java.class.path	NULL	PHP_INI_ALL
java.home	NULL	PHP_INI_ALL
java.library.path	NULL	PHP_INI_ALL
java.library	JAVALIB	PHP_INI_ALL
java.library	NULL	PHP_INI_ALL
ldap.max_links	"-1"	PHP_INI_SYSTEM
mbstring.detect_order	NULL	PHP_INI_ALL
mbstring.http_input	NULL	PHP_INI_ALL
mbstring.http_output	NULL	PHP_INI_ALL
mbstring.internal_encoding	NULL	PHP_INI_ALL
mbstring.substitute_character	NULL	PHP_INI_ALL
mbstring.func_overload	"0"	PHP_INI_SYSTEM
mcrypt.algorithms_dir	NULL	PHP_INI_ALL
mcrypt.modes_dir	NULL	PHP_INI_ALL
mime_magic.magicfile	"/usr/share/misc/magic.mime"	PHP_INI_SYSTEM
mssql.allow_persistent	"1"	PHP_INI_SYSTEM
mssql.max_persistent	"-1"	PHP_INI_SYSTEM
mssql.max_links	"-1"	PHP_INI_SYSTEM
mssql.min_error_severity	"10"	PHP_INI_ALL
mssql.min_message_severity	"10"	PHP_INI_ALL
mssql.compatibility_mode	"0"	PHP_INI_ALL
mssql.connect_timeout	"5"	PHP_INI_ALL
mssql.timeout	"60"	PHP_INI_ALL
mssql.textsize	"-1"	PHP_INI_ALL
mssql.textlimit	"-1"	PHP_INI_ALL

Name	Default	Changeable
mssql.batchsize	"0"	PHP_INI_ALL
mssql.datetimeconvert	"1"	PHP_INI_ALL
mysql.allow_persistent	"1"	PHP_INI_SYSTEM
mysql.max_persistent	"-1"	PHP_INI_SYSTEM
mysql.max_links	"-1"	PHP_INI_SYSTEM
mysql.default_host	NULL	PHP_INI_ALL
mysql.default_user	NULL	PHP_INI_ALL
mysql.default_password	NULL	PHP_INI_ALL
mysql.default_port	NULL	PHP_INI_ALL
mysql.default_socket	NULL	PHP_INI_ALL
ncurses.value	"42"	PHP_INI_ALL
ncurses.string	"foobar"	PHP_INI_ALL
odbc.allow_persistent	"1"	PHP_INI_SYSTEM
odbc.max_persistent	"-1"	PHP_INI_SYSTEM
odbc.max_links	"-1"	PHP_INI_SYSTEM
odbc.default_db	NULL	PHP_INI_ALL
odbc.default_user	NULL	PHP_INI_ALL
odbc.default_pw	NULL	PHP_INI_ALL
odbc.defaultlrl	"4096"	PHP_INI_ALL
odbc.defaultbinmode	"1"	PHP_INI_ALL
odbc.check_persistent	"1"	PHP_INI_SYSTEM
pfpro.defaultthost	"test.signio.com"	
pfpro.defaultthost	"test-payflow.verisign.com"	
pfpro.defaultport	"443"	PHP_INI_ALL
pfpro.defaulttimeout	"30"	PHP_INI_ALL
pfpro.proxyaddress	""	PHP_INI_ALL
pfpro.proxyport	""	PHP_INI_ALL
pfpro.proxylogon	""	PHP_INI_ALL
pfpro.proxypassword	""	PHP_INI_ALL
pgsql.allow_persistent	"1"	PHP_INI_SYSTEM
pgsql.max_persistent	"-1"	PHP_INI_SYSTEM
pgsql.max_links	"-1"	PHP_INI_SYSTEM
pgsql.auto_reset_persistent	"0"	PHP_INI_SYSTEM
pgsql.ignore_notice	"0"	PHP_INI_ALL
pgsql.log_notice	"0"	PHP_INI_ALL
session.save_path	"/tmp"	PHP_INI_ALL
session.name	"PHPSESSID"	PHP_INI_ALL
session.save_handler	"files"	PHP_INI_ALL

Name	Default	Changeable
session.auto_start	"0"	PHP_INI_ALL
session.gc_probability	"1"	PHP_INI_ALL
session.gc_maxlifetime	"1440"	PHP_INI_ALL
session.serialize_handler	"php"	PHP_INI_ALL
session.cookie_lifetime	"0"	PHP_INI_ALL
session.cookie_path	"/"	PHP_INI_ALL
session.cookie_domain	""	PHP_INI_ALL
session.cookie_secure	""	PHP_INI_ALL
session.use_cookies	"1"	PHP_INI_ALL
session.referer_check	""	PHP_INI_ALL
session.entropy_file	""	PHP_INI_ALL
session.entropy_length	"0"	PHP_INI_ALL
session.cache_limiter	"nocache"	PHP_INI_ALL
session.cache_expire	"180"	PHP_INI_ALL
session.use_trans_sid	"1"	PHP_INI_ALL
session.encode_sources	"globals"	track"
extname.global_value	"42"	PHP_INI_ALL
extname.global_string	"foobar"	PHP_INI_ALL
assert.active	"1"	PHP_INI_ALL
assert.bail	"0"	PHP_INI_ALL
assert.warning	"1"	PHP_INI_ALL
assert.callback	NULL	PHP_INI_ALL
assert.quiet_eval	"0"	PHP_INI_ALL
safe_mode_protected_env_vars	SAFE_MODE_PROTECTED_ENV_VARS	PHP_INI_SYSTEM
safe_mode_allowed_env_vars	SAFE_MODE_ALLOWED_ENV_VARS	PHP_INI_SYSTEM
url_rewriter.tags	"a=href"	area=href
url_rewriter.tags	"a=href"	area=href
sybct.allow_persistent	"1"	PHP_INI_SYSTEM
sybct.max_persistent	"-1"	PHP_INI_SYSTEM
sybct.max_links	"-1"	PHP_INI_SYSTEM
sybct.min_server_severity	"10"	PHP_INI_ALL
sybct.min_client_severity	"10"	PHP_INI_ALL
sybct.hostname	NULL	PHP_INI_ALL
tokenizer.global_value	"42"	PHP_INI_ALL
tokenizer.global_string	"foobar"	PHP_INI_ALL
vpopmail.directory	""	PHP_INI_ALL
zlib.output_compression	"0"	PHP_INI_ALL

Name	Default	Changeable
zlib.output_compression_level	"-1"	PHP_INI_ALL
define_syslog_variables	"0"	PHP_INI_ALL
highlight.bg	HL_BG_COLOR	PHP_INI_ALL
highlight.comment	HL_COMMENT_COLOR	PHP_INI_ALL
highlight.default	HL_DEFAULT_COLOR	PHP_INI_ALL
highlight.html	HL_HTML_COLOR	PHP_INI_ALL
highlight.keyword	HL_KEYWORD_COLOR	PHP_INI_ALL
highlight.string	HL_STRING_COLOR	PHP_INI_ALL
allow_call_time_pass_reference	"1"	PHP_INI_SYSTEM PHP_INI_PERDIR
asp_tags	"0"	PHP_INI_SYSTEM PHP_INI_PERDIR
display_errors	"1"	PHP_INI_ALL
display_startup_errors	"0"	PHP_INI_ALL
enable_dl	"1"	PHP_INI_SYSTEM
expose_php	"1"	PHP_INI_SYSTEM
html_errors	"1"	PHP_INI_SYSTEM
xmlrpc_errors	"0"	PHP_INI_SYSTEM
xmlrpc_error_number	"0"	PHP_INI_ALL
ignore_user_abort	"0"	PHP_INI_ALL
implicit_flush	"0"	PHP_INI_PERDIR PHP_INI_SYSTEM
log_errors	"0"	PHP_INI_ALL
log_errors_max_len	"1024"	PHP_INI_ALL
ignore_repeated_errors	"0"	PHP_INI_ALL
ignore_repeated_source	"0"	PHP_INI_ALL
magic_quotes_gpc	"1"	PHP_INI_ALL
magic_quotes_runtime	"0"	PHP_INI_ALL
magic_quotes_sybase	"0"	PHP_INI_ALL
output_buffering	"0"	PHP_INI_PERDIR PHP_INI_SYSTEM
output_handler	NULL	PHP_INI_PERDIR PHP_INI_SYSTEM
register_argc_argv	"1"	PHP_INI_ALL
register_globals	"0"	PHP_INI_ALL
safe_mode	"1"	PHP_INI_SYSTEM
safe_mode	"0"	PHP_INI_SYSTEM
safe_mode_include_dir	NULL	PHP_INI_SYSTEM
safe_mode_gid	"0"	PHP_INI_SYSTEM

Name	Default	Changeable
short_open_tag	DEFAULT_SHORT_OPEN_TAG	PHP_INI_SYSTEM PHP_INI_PERDIR
sql.safe_mode	"0"	PHP_INI_SYSTEM
track_errors	"0"	PHP_INI_ALL
y2k_compliance	"0"	PHP_INI_ALL
unserialize_callback_func	NULL	PHP_INI_ALL
arg_separator.output	"&"	PHP_INI_ALL
arg_separator.input	"&"	PHP_INI_SYSTEM PHP_INI_PERDIR
auto_append_file	NULL	PHP_INI_ALL
auto_prepend_file	NULL	PHP_INI_ALL
doc_root	NULL	PHP_INI_SYSTEM
default_charset	SAPI_DEFAULT_CHARSET	PHP_INI_ALL
default_mimetype	SAPI_DEFAULT_MIMETYPE	PHP_INI_ALL
error_log	NULL	PHP_INI_ALL
extension_dir	PHP_EXTENSION_DIR	PHP_INI_SYSTEM
gpc_order	"GPC"	PHP_INI_ALL
include_path	PHP_INCLUDE_PATH	PHP_INI_ALL
max_execution_time	"30"	PHP_INI_ALL
open_basedir	NULL	PHP_INI_SYSTEM
safe_mode_exec_dir	"1"	PHP_INI_SYSTEM
upload_max_filesize	"2M"	PHP_INI_SYSTEM
file_uploads	"1"	PHP_INI_ALL
post_max_size	"8M"	PHP_INI_SYSTEM
upload_tmp_dir	NULL	PHP_INI_SYSTEM
user_dir	NULL	PHP_INI_SYSTEM
variables_order	NULL	PHP_INI_ALL
error_append_string	NULL	PHP_INI_ALL
error_prepend_string	NULL	PHP_INI_ALL
SMTP	"localhost"	PHP_INI_ALL
browscap	NULL	PHP_INI_SYSTEM
error_reporting	NULL	PHP_INI_ALL
memory_limit	"8M"	PHP_INI_ALL
precision	"14"	PHP_INI_ALL
sendmail_from	NULL	PHP_INI_ALL
sendmail_path	DEFAULT_SENDMAIL_PATH	PHP_INI_SYSTEM
disable_functions	""	PHP_INI_SYSTEM
allow_url_fopen	"1"	PHP_INI_ALL
always_populate_raw_post_data	"0"	PHP_INI_ALL

Name	Default	Changeable
xbithack	"0"	PHP_INI_ALL
engine	"1"	PHP_INI_ALL
last_modified	"0"	PHP_INI_ALL
child_terminate	"0"	PHP_INI_ALL
async_send	"0"	PHP_INI_ALL

Table 2. Definition of PHP_INI_* constants

Constant	Value	Meaning
PHP_INI_USER	1	Entry can be set in user scripts
PHP_INI_PERDIR	2	Entry can be set in .htaccess
PHP_INI_SYSTEM	4	Entry can be set in php.ini or httpd.conf
PHP_INI_ALL	7	Entry can be set anywhere

See also: `ini_alter()`, `ini_get()`, and `ini_restore()`

php_logo_guid (PHP 4 >= 4.0.0)

Gets the logo guid

```
string php_logo_guid ( void) \linebreak
```

Note: This functionality was added in PHP 4.0.0.

See also: `phpinfo()`, `phpversion()`, and `phpcredits()`.

php_sapi_name (PHP 4)

Returns the type of interface between web server and PHP

```
string php_sapi_name ( void) \linebreak
```

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) \linebreak

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");
}

```

phpcredits (PHP 4 >= 4.0.0)

Prints out the credits for PHP

void **phpcredits** ([int flag]) \linebreak

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. *flag* is optional, and it defaults to `CREDITS_ALL`. To generate a custom credits page, you may want to use the *flag* parameter. 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);
// 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
CREDITS_FULLPAGE	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.

name	description
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()`, and `php_logo_guid()`.

phpinfo (PHP 3, PHP 4 >= 4.0.0)

Outputs lots of PHP information

int **phpinfo** ([int *what*]) \linebreak

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.

Because every system is setup differently, **phpinfo()** is commonly used to check configuration settings and for available predefined variables on a given system. Also, **phpinfo()** is a valuable debugging tool as it contains all EGPCS (Environment, GET, POST, Cookie, Server) data.

The output may be customized by passing one or more of the following *constants* bitwise values summed together in the optional *what* parameter. One can also combine the respective constants or bitwise values together with the or operator.

Table 1. phpinfo() options

Name (constant)	Value	Description
INFO_GENERAL	1	The configuration line, <code>php.ini</code> location, build date, Web Server, System and more.
INFO_CREDITS	2	PHP 4 Credits. See also <code>phpcredits()</code> .
INFO_CONFIGURATION	4	Current Local and Master values for <code>php</code> directives. See also <code>ini_get()</code> .
INFO_MODULES	8	Loaded modules and their respective settings.
INFO_ENVIRONMENT	16	Environment Variable information that's also available in <code>\$_ENV</code> .

Name (constant)	Value	Description
INFO_VARIABLES	32	Shows all predefined variables from EGPCS (Environment, GET, POST, Cookie, Server).
INFO_LICENSE	64	PHP License information. See also the license faq (http://www.php.net/license/).
INFO_ALL	-1	Shows all of the above. This is the default value.

Example 1. phpinfo() examples

```
<?php

// Show all information, defaults to INFO_ALL
phpinfo();

// Show just the module information.
// phpinfo(8) yields identical results.
phpinfo(INFO_MODULES);

?>
```

Note: Parts of the information displayed are disabled when the `expose_php` configuration setting is set to `off`. This includes the PHP and Zend logos, and the credits.

See also: `phpversion()`, `phpcredits()`, `php_logo_guid()`, `ini_get()`, `ini_set()`, and the section on Predefined Variables.

phpversion (PHP 3, PHP 4 >= 4.0.0)

Gets the current PHP version

string **phpversion** (void) \linebreak

Returns a string containing the version of the currently running PHP parser.

Note: This information is also available in the predefined constant `PHP_VERSION`.

Example 1. phpversion() Example

```
<?php
// prints e.g. 'Current PHP version: 4.1.1'
echo 'Current PHP version: ' . phpversion();
?>
```

See also `version_compare()`, `phpinfo()`, `phpcredits()`, `php_logo_guid()`, and `zend_version()`.

putenv (PHP 3, PHP 4 >= 4.0.0)

Sets the value of an environment variable

```
void putenv ( string setting ) \linebreak
```

Adds *setting* to the server environment. The environment variable will only exist for the duration of the current request. At the end of the request the environment is restored to its original state.

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

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

Warning

These directives have only effect when safe-mode itself is enabled!

Example 1. Setting an Environment Variable

```
putenv ("UNIQID=$uniqid");
```

See also `getenv()`.

set_magic_quotes_runtime (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Sets the current active configuration setting of `magic_quotes_runtime`

```
long set_magic_quotes_runtime ( int new_setting) \linebreak
```

Set the current active configuration setting of `magic_quotes_runtime` (0 for off, 1 for on).

See also: `get_magic_quotes_gpc()` and `get_magic_quotes_runtime()`.

set_time_limit (PHP 3, PHP 4 >= 4.0.0)

Limits the maximum execution time

```
void set_time_limit ( int seconds) \linebreak
```

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.

`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.

Note: The `set_time_limit()` function and the configuration directive `max_execution_time` only affect the execution time of the script itself. Any time spent on activity that happens outside the execution of the script such as system calls using `system()`, the `sleep()` function, database queries, etc. is not included when determining the maximum time that the script has been running.

version_compare (PHP 4 >= 4.1.0)

Compares two "PHP-standardized" version number strings

```
int version_compare ( string version1, string version2 [, string operator]) \linebreak
```

`version_compare()` compares two "PHP-standardized" version number strings. This is useful if you would like to write programs working only on some versions of PHP.

version_compare() returns -1 if the first version is lower than the second, 0 if they are equal, and +1 if the second is lower.

If you specify the third optional *operator* argument, you can test for a particular relationship. The possible operators are: <, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne respectively. Using this argument, the function will return 1 if the relationship is the one specified by the operator, 0 otherwise.

Example 1. version_compare() Example

```
// prints -1
echo version_compare("4.0.4", "4.0.6");

// these all print 1
echo version_compare("4.0.4", "4.0.6", "<");
echo version_compare("4.0.6", "4.0.6", "eq");
```

zend_logo_guid (PHP 4 >= 4.0.0)

Gets the zend guid

string **zend_logo_guid** (void) \linebreak

Note: This functionality was added in PHP 4.0.0.

zend_version (PHP 4 >= 4.0.0)

Gets the version of the current Zend engine

string **zend_version** (void) \linebreak

Returns a string containing the version of the currently running PHP parser.

Example 1. zend_version() Example

```
// prints e.g. 'Zend engine version: 1.0.4'
echo "Zend engine version: " . zend_version();
```

See also `phpinfo()`, `phpcredits()`, `php_logo_guid()`, and `phpversion()`.

LXXVIII. 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_ctermid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Get path name of controlling terminal

string **posix_ctermid** (void) \linebreak

Needs to be written.

posix_getcwd (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Pathname of current directory

string **posix_getcwd** (void) \linebreak

Needs to be written ASAP.

posix_getegid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the effective group ID of the current process

int **posix_getegid** (void) \linebreak

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_geteuid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the effective user ID of the current process

int **posix_geteuid** (void) \linebreak

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 >= 4.0.0)

Return the real group ID of the current process

int **posix_getgid** (void) \linebreak

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_getgrgid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return info about a group by group id

array **posix_getgrgid** (int gid) \linebreak

Needs to be written.

posix_getgrnam (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return info about a group by name

array **posix_getgrnam** (string name) \linebreak

Needs to be written.

posix_getgroups (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the group set of the current process

array **posix_getgroups** (void) \linebreak

Returns an array of integers containing the numeric group ids of the group set of the current process. See also `posix_getgrgid()` for information on how to convert this into useable group names.

posix_getlogin (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return login name

string **posix_getlogin** (void) \linebreak

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_getpgid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Get process group id for job control

int **posix_getpgid** (int pid) \linebreak

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_getpgrp (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the current process group identifier

```
int posix_getpgrp ( void) \linebreak
```

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_getpid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the current process identifier

```
int posix_getpid ( void) \linebreak
```

Return the process identifier of the current process.

posix_getppid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the parent process identifier

```
int posix_getppid ( void) \linebreak
```

Return the process identifier of the parent process of the current process.

posix_getpwnam (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Return info about a user by username

```
array posix_getpwnam ( string username) \linebreak
```

Returns an associative array containing information about a user referenced by an alphanumeric username, passed in the `username` parameter.

The array elements returned are:

Element	Description
---------	-------------

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 <code>posix_getgrgid()</code> 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_getpwuid (PHP 3 >= 3.0.13, PHP 4 >= 4.0.0)

Return info about a user by user id

array **posix_getpwuid** (int uid) \linebreak

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 <code>posix_getgrgid()</code> 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 >= 4.0.0)

Return info about system resource limits

array **posix_getrlimit** (void) \linebreak

Needs to be written ASAP.

posix_getsid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Get the current sid of the process

int **posix_getsid** (int pid) \linebreak

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_getuid (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Return the real user ID of the current process

int **posix_getuid** (void) \linebreak

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_isatty (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Determine if a file descriptor is an interactive terminal

bool **posix_isatty** (int fd) \linebreak

Needs to be written.

posix_kill (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Send a signal to a process

bool **posix_kill** (int pid, int sig) \linebreak

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_mkfifo (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Create a fifo special file (a named pipe)

bool **posix_mkfifo** (string pathname, int mode) \linebreak

posix_mkfifo() creates a special `FIFO` file which exists in the file system and acts as a bidirectional communication endpoint for processes.

The second parameter *mode* has to be given in octal notation (e.g. 0644). The permission of the newly created FIFO also depends on the setting of the current `umask()`. The permissions of the created file are `(mode & ~umask)`.

Note: When safe mode is enabled, PHP checks whether the directory in which you are about to operate has the same UID as the script that is being executed.

posix_setegid (PHP 4 >= 4.0.2)

Set the effective GID of the current process

```
bool posix_setegid ( int gid) \linebreak
```

Set the effective 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.

Returns `TRUE` on success, `FALSE` otherwise.

posix_seteuid (PHP 4 >= 4.0.2)

Set the effective UID of the current process

```
bool posix_seteuid ( int uid) \linebreak
```

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 >= 4.0.0)

Set the GID of the current process

```
bool posix_setgid ( int gid) \linebreak
```

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_setpgid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

set process group id for job control

int **posix_setpgid** (int pid, int pgid) \linebreak

Let the process *pid* join the process group *pgid*. See POSIX.1 and the `setuid(2)` manual page on your POSIX system for more informations on process groups and job control. Returns `TRUE` on success, `FALSE` otherwise.

posix_setsid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Make the current process a session leader

int **posix_setsid** (void) \linebreak

Make the current process a session leader. See POSIX.1 and the `setuid(2)` manual page on your POSIX system for more informations on process groups and job control. Returns the session id.

posix_setuid (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Set the UID of the current process

bool **posix_setuid** (int uid) \linebreak

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_times (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Get process times

array **posix_times** (void) \linebreak

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_ttyname (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Determine terminal device name

string **posix_ttyname** (int fd) \linebreak

Needs to be written.

posix_uname (PHP 3>= 3.0.10, PHP 4 >= 4.0.0)

Get system name

array **posix_uname** (void) \linebreak

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.

LXXIX. 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/SQL99 language support, transaction integrity and type extensibility. PostgreSQL is an open source descendant of this original Berkeley code.

PostgreSQL database is Open Source product and available without cost. To use PostgreSQL support, you need PostgreSQL 6.5 or later. PostgreSQL 7.0 or later to enable all PostgreSQL module feature. PostgreSQL supports many character encoding including multibyte character encoding. The current version and more information about PostgreSQL is available at <http://www.postgresql.org/>.

In order to enable PostgreSQL support, `--with-pgsql[=DIR]` is required when you compile PHP. If shared object module is available, PostgreSQL module may be loaded using extension directive in `php.ini` or `dl()` function. Supported ini directives are described in `php.ini-dist` which comes with

source distribution.

Warning

Using the PostgreSQL module with PHP 4.0.6 is not recommended due to a bug in the notice message handling code. Use 4.1.0 or later.

Warning

PostgreSQL function names will be changed in 4.2.0 release to conform to current coding standards. Most of new names will have additional underscores, e.g. `pg_lo_open()`. Some functions are renamed to different name for consistency. e.g. `pg_exec()` to `pg_query()`. Older names can be used in 4.2.0 and a few releases from 4.2.0, but they may be deleted in the future.

Table 1. Function names changed

Old name	New name
<code>pg_exec()</code>	<code>pg_query()</code>
<code>pg_getlastoid()</code>	<code>pg_last_oid()</code>
<code>pg_cmdtuples()</code>	<code>pg_affected_rows()</code>
<code>pg_numrows()</code>	<code>pg_num_rows()</code>
<code>pg_numfields()</code>	<code>pg_num_fields()</code>
<code>pg_fieldname()</code>	<code>pg_field_name()</code>
<code>pg_fieldsize()</code>	<code>pg_field_size()</code>
<code>pg_fieldnum()</code>	<code>pg_field_num()</code>
<code>pg_fieldprtlen()</code>	<code>pg_field prtlen()</code>
<code>pg_fieldisnull()</code>	<code>pg_field_is_null()</code>
<code>pg_freeresult()</code>	<code>pg_free_result()</code>
<code>pg_result()</code>	<code>pg_fetch_result()</code>
<code>pg_loreadall()</code>	<code>pg_lo_read_all()</code>
<code>pg_locreate()</code>	<code>pg_lo_create()</code>
<code>pg_lounlink()</code>	<code>pg_lo_unlink()</code>
<code>pg_loopen()</code>	<code>pg_lo_unlink()</code>
<code>pg_loclose()</code>	<code>pg_lo_close()</code>
<code>pg_loread()</code>	<code>pg_lo_read()</code>
<code>pg_lowrite()</code>	<code>pg_lo_write()</code>
<code>pg_loimport()</code>	<code>pg_lo_import()</code>
<code>pg_loexport()</code>	<code>pg_lo_export()</code>

The old `pg_connect()/pg_pconnect()` syntax will be deprecated to support asynchronous connections in the future. Please use a connection string for `pg_connect()` and `pg_pconnect()`.

Not all functions are supported by all builds. It depends on your libpq (The PostgreSQL C Client

interface) version and how libpq is compiled. If there is missing function, libpq does not support the feature required for the function.

It is also important that you use newer libpq than PostgreSQL Server to be connected. If you use libpq older than PostgreSQL Server expects, you may have problems.

Since version 6.3 (03/02/1998) PostgreSQL uses unix domain sockets by default. TCP port will NOT be opened by default. 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 2. 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.php on line 20.
postmaster -i &	<code>pg_connect("host=localhost dbname=MyDbName");</code>	OK

A connection to PostgreSQL server can be established with the following value pairs set in the command string: **`$conn = pg_connect("host=myHost port=myPort tty=myTTY options=myOptions dbname=myDB user=myUser password=myPassword ");`**

The previous syntax of: **`$conn = pg_connect ("host", "port", "options", "tty", "dbname")`** has been deprecated.

Environmental variables affect PostgreSQL server/client behavior. For example, PostgreSQL module will lookup PGHOST environment variable when the hostname is omitted in the connection string.

Supported environment variables are different from version to version. Refer to PostgreSQL Programmer's Manual (libpq - Environment Variables) for details.

Make sure you set environment variables for appropriate user. Use `$_ENV` or `getenv()` to check which environment variables are available to the current process.

Example 1. Setting default parameters

```
PGHOST=pgsql.example.com
PGPORT=7890
PGDATABASE=web-system
PGUSER=web-user
PGPASSWORD=secret
```

```
PGDATESTYLE=ISO
PGTZ=JST
PGCLIENTENCODING=EUC-JP
```

```
export PGHOST PGPORT PGDATABASE PGUSER PGPASSWORD PGDATESTYLE PGTZ PGCLIENTENCODING
```

Starting with PostgreSQL 7.1.0, you can store up to 1GB into a field of type text. In older versions, this was limited to the block size (default was 8KB, maximum was 32KB, defined at compile time)

To use the large object (lo) interface, it is required to enclose large object functions within a transaction block. A transaction block starts with a SQL statement **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 2. Using Large Objects

```
<?php
    $database = pg_connect ("dbname=jakarta");
    pg_query ($database, "begin");
    $oid = pg_lo_create ($database);
    echo "$oid\n";
    $handle = pg_lo_open ($database, $oid, "w");
    echo "$handle\n";
    pg_lo_write ($handle, "large object data");
    pg_lo_close ($handle);
    pg_query ($database, "commit");
?>
```

You should not close the connection to the PostgreSQL server before closing the large object.

pg_affected_rows (PHP 4 >= 4.2.0)

Returns number of affected records(tuples)

int **pg_affected_rows** (resource result) \linebreak

pg_affected_rows() returns the number of tuples (instances/records/rows) affected by INSERT, UPDATE, and DELETE queries executed by `pg_query()`. If no tuple is affected by this function, it will return 0.

Example 1. pg_affected_rows()

```
<?php
    $result = pg_query ($conn, "INSERT INTO publisher VALUES ('Author')");
    $cmdtuples = pg_affected_rows ($result);
    echo $cmdtuples . " tuples are affected.";
?>
```

Note: This function used to be called `pg_cmdtuples()`.

See also `pg_query()` and `pg_num_rows()`.

pg_cancel_query (PHP 4 >= 4.2.0)

Cancel async query

bool **pg_cancel_query** (resource connection) \linebreak

pg_cancel_query() cancel asynchronous query sent by `pg_send_query()`. You cannot cancel query executed by `pg_query()`.

See also `pg_send_query()` and `pg_connection_busy()`

pg_client_encoding (PHP 3 CVS only, PHP 4 >= 4.0.3)

Get the client encoding

string **pg_client_encoding** ([resource connection]) \linebreak

pg_client_encoding() 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-4.0.3 or higher and PostgreSQL-7.0 or higher. If libpq is compiled without multibyte encoding support, pg_set_client_encoding() always return "SQL_ASCII". Supported encoding depends on PostgreSQL version. Refer to PostgreSQL manual for details to enable multibyte support and encoding supported.

The function used to be called **pg_clientencoding()**.

See also pg_set_client_encoding().

pg_close (PHP 3, PHP 4 >= 4.0.0)

Close a PostgreSQL connection

bool **pg_close** (resource connection) \linebreak

pg_close() closes the non-persistent connection to a PostgreSQL database associated with the given *connection* resource. Returns TRUE on success, FALSE on failure.

Note: Using **pg_close()** is not usually necessary, as non-persistent open connections are automatically closed at the end of the script.

If there is open large object resource on the connection, do not close the connection before closing all large object resources.

pg_connect (PHP 3, PHP 4 >= 4.0.0)

Open a PostgreSQL connection

resource **pg_connect** (string connection_string) \linebreak

pg_connect() returns a connection resource that is needed by other PostgreSQL functions.

pg_connect() opens a connection to a PostgreSQL database specified by the *connection_string*. It returns a connection resource on success. It returns FALSE if the connection could not be made. *connection_string* should be a quoted string.

Example 1. Using pg_connect

```
<?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 ("host=sheep port=5432 dbname=mary user=lamb password=foo");
//connect to a database named "mary" on the host "sheep" with a username and password
$conn_string = "host=sheep port=5432 dbname=test user=lamb password=bar";
$dbconn4 = pg_connect ($conn_string);
//connect to a database named "test" on the host "sheep" with a username and password
?>

```

The arguments available for *connection_string* includes *host*, *port*, *tty*, *options*, *dbname*, *user*, and *password*.

If a second call is made to **pg_connect()** with the same *connection_string*, no new connection will be established, but instead, the connection resource of the already opened connection will be returned. You can have multiple connections to the same database if you use different connection string.

The old syntax with multiple parameters **\$conn = pg_connect ("host", "port", "options", "tty", "dbname")** has been deprecated.

See also `pg_pconnect()`, `pg_close()`, `pg_host()`, `pg_port()`, `pg_tty()`, `pg_options()` and `pg_dbname()`.

pg_connection_busy (PHP 4 >= 4.2.0)

Get connection is busy or not

```
bool pg_connection_busy ( resource connection) \linebreak
```

pg_connection_busy() returns `TRUE` if the connection is busy. If it is busy, a previous query is still executing. If `pg_get_result()` is called, it will be blocked.

See also `pg_connection_status()` and `pg_get_result()`

pg_connection_reset (PHP 4 >= 4.2.0)

Reset connection (reconnect)

```
bool pg_connection_reset ( resource connection) \linebreak
```

pg_connection_reset() resets the connection. It is useful for error recovery. Returns `TRUE` on success, `FALSE` on failure.

See also `pg_connect()`, `pg_pconnect()` and `pg_connection_status()`

pg_connection_status (PHP 4 >= 4.2.0)

Get connection status

```
int pg_connection_status ( resource connection) \linebreak
```

pg_connection_status() returns a connection status. Possible statuses are `PGSQL_CONNECTION_OK` and `PGSQL_CONNECTION_BAD`.

See also `pg_connection_busy()`.

pg_convert (PHP 4 CVS only)

Convert associative array value into suitable for SQL statement.

```
array pg_convert ( resource connection, string table_name, array assoc_array [, int options]) \linebreak
```

pg_convert() check and convert `assoc_array` suitable for SQL statement.

Note: This function is experimental.

See also `pg_metadata()`

pg_copy_from (PHP 4 >= 4.2.0)

Insert records into a table from an array

```
int pg_copy_from ( int connection, string table_name, array rows [, string delimiter [, string null_as]]) \linebreak
```

pg_copy_from() insert records into a table from `rows`. It issues `COPY` command internally to insert records. Returns `TRUE` on success, `FALSE` on failure.

See also `pg_copy_to()`

pg_copy_to (PHP 4 >= 4.2.0)

Copy a table to an array

```
int pg_copy_to ( int connection, string table_name [, string delimiter [, string null_as]]) \linebreak
```

pg_copy_to() copies a table to an array. The resulting array is returned. It returns `FALSE` on failure.

See also `pg_copy_from()`

pg_dbname (PHP 3, PHP 4 >= 4.0.0)

Get the database name

string **pg_dbname** (resource connection) \linebreak

pg_dbname() returns the name of the database that the given PostgreSQL *connection* resource. It returns *FALSE*, if *connection* is not a valid PostgreSQL connection resource.

pg_delete (PHP 4 CVS only)

Delete records.

long **pg_delete** (resource connection, string table_name, array assoc_array [, int options]) \linebreak

pg_delete() deletes record condition specified by *assoc_array* which has *field=>value*. If *option* is specified, *pg_convert()* is applied to *assoc_array* with specified option.

Example 1. pg_delete

```

<?php
    $db = pg_connect ('dbname=foo');
    // This is safe, since $_POST is converted automatically
    $res = pg_delete($db, 'post_log', $_POST);
    if ($res) {
        echo "POST data is deleted: $res\n";
    }
    else {
        echo "User must have sent wrong inputs\n";
    }
?>

```

Note: This function is experimental.

See also *pg_convert()*

pg_end_copy (PHP 4 >= 4.0.3)

Sync with PostgreSQL backend

bool **pg_end_copy** ([resource connection]) \linebreak

pg_end_copy() syncs the PostgreSQL frontend (usually a web server process) with the PostgreSQL server after doing a copy operation performed by `pg_put_line()`. **pg_end_copy()** must be issued, otherwise the PostgreSQL server may get out of sync with the frontend and will report an error. Returns `TRUE` on success, `FALSE` on failure.

For further details and an example, see also `pg_put_line()`.

pg_escape_bytea (PHP 4 >= 4.2.0)

Escape binary for bytea type

string **pg_escape_bytea** (string data) \linebreak

pg_escape_bytea() escapes string for bytea datatype. It returns escaped string.

Note: When you SELECT bytea type, PostgreSQL returns octal byte value prefixed by \. (e.g. \032) Users are supposed to convert back to binary format by yourself.

This function requires PostgreSQL 7.2 or later. With PostgreSQL 7.2.0 and 7.2.1, bytea type must be casted when you enable multi-byte support. i.e. `INSERT INTO test_table (image) VALUES ('$image_escaped'::bytea);` PostgreSQL 7.2.2 or later does not need cast. Exception is when client and backend character encoding does not match, there may be multi-byte stream error. User must cast to bytea to avoid this error.

Newer PostgreSQL will support unescape function. Support for built-in unescape function will be added when it's available.

See also `pg_escape_string()`

pg_escape_string (PHP 4 >= 4.2.0)

Escape string for text/char type

string **pg_escape_string** (string data) \linebreak

pg_escape_string() escapes string for text/char datatype. It returns escaped string for PostgreSQL. Use of this function is recommended instead of `addslashes()`.

Note: This function requires PostgreSQL 7.2 or later.

See also `pg_escape_bytea()`

pg_fetch_array (PHP 3 >= 3.0.1, PHP 4 >= 4.0.0)

Fetch a row as an array

array **pg_fetch_array** (resource result, int row [, int result_type]) \linebreak

pg_fetch_array() returns an array that corresponds to the fetched row (tuples/records). It returns `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 (field index) to the result array, it also stores the data in associative indices (field name) by default.

row is row (record) number to be retrieved. First row is 0.

result_type is optional parameter controls how return value is initialized. *result_type* is a constant and can take the following values: `PGSQL_ASSOC`, `PGSQL_NUM`, and `PGSQL_BOTH`.

pg_fetch_array() returns associative array that has field name as key for `PGSQL_ASSOC`. field index as key with `PGSQL_NUM` and both field name/index as key with `PGSQL_BOTH`. Default is `PGSQL_BOTH`.

Note: *result_type* was added in PHP 4.0.

pg_fetch_array() is NOT significantly slower than using `pg_fetch_row()`, while it provides a significant ease of use.

See also `pg_fetch_row()` and `pg_fetch_object()` and `pg_fetch_result()`.

Example 1. PostgreSQL fetch array

```
<?php
$conn = pg_pconnect ("dbname=publisher");
if (!$conn) {
    echo "An error occured.\n";
    exit;
}

$result = pg_query ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occured.\n";
    exit;
}

$arr = pg_fetch_array ($result, 0, PGSQL_NUM);
echo $arr[0] . " <- array\n";

$arr = pg_fetch_array ($result, 1, PGSQL_ASSOC);
echo $arr["author"] . " <- array\n";
?>
```

Note: From 4.1.0, *row* became optional. Calling **pg_fetch_array()** will increment internal row counter by 1.

pg_fetch_object (PHP 3 >= 3.0.1, PHP 4 >= 4.0.0)

Fetch a row as an object

object **pg_fetch_object** (resource result, int row [, int result_type]) \linebreak

pg_fetch_object() returns an object with properties that correspond to the fetched row. It returns `FALSE` if there are no more rows or error.

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).

row is row (record) number to be retrieved. First row is 0.

Speed-wise, the function is identical to `pg_fetch_array()`, and almost as quick as `pg_fetch_row()` (the difference is insignificant).

Note: Form 4.3.0, *result_type* is default to `PGSQL_ASSOC` while older versions' default was `PGSQL_BOTH`. There is no use for numeric property, since numeric property name is invalid in PHP.

result_type may be deleted in future versions.

See also `pg_query()`, `pg_fetch_array()`, `pg_fetch_row()` and `pg_fetch_result()`.

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;

$qu = pg_query ($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++;
}
?>
<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;
    $row++;
}
echo "-----\n";
?>
</PRE>
<?php
pg_free_result ($qu);
pg_close ($db_conn);
?>

```

Note: From 4.1.0, *row* became optional. Calling **pg_fetch_object()** will increment internal row counter counter by 1.

pg_fetch_result (PHP 4 >= 4.2.0)

Returns values from a result resource

mixed **pg_fetch_result** (resource result, int row, mixed field) \linebreak

pg_fetch_result() returns values from a *result* resource returned by `pg_query()`. *row* is integer. *field* is field name(string) or field index (integer). The *row* and *field* 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 void types are returned as integer values. All forms of float, and real types are returned as float 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_fetch_row (PHP 3 >= 3.0.1, PHP 4 >= 4.0.0)

Get a row as an enumerated array

array **pg_fetch_row** (resource result, int row) \linebreak

pg_fetch_row() fetches one row of data from the result associated with the specified *result* resource. The row (record) is returned as an array. Each result column is stored in an array offset, starting at offset 0.

It returns an array that corresponds to the fetched row, or FALSE if there are no more rows.

See also: `pg_query()`, `pg_fetch_array()`, `pg_fetch_object()` and `pg_fetch_result()`.

Example 1. Postgres fetch row

```
<?php
$conn = pg_pconnect ("dbname=publisher");
if (!$conn) {
    echo "An error occured.\n";
    exit;
}

$result = pg_query ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occured.\n";
    exit;
}

$num = pg_num_rows($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>";
}

?>
```

Note: From 4.1.0, *row* became optional. Calling **pg_fetch_row()** will increment internal row counter by 1.

pg_field_is_null (PHP 4 >= 4.2.0)

Test if a field is `NULL`

```
int pg_field_is_null ( resource result, int row, mixed field) \linebreak
```

pg_field_is_null() test if a field is `NULL` or not. It returns 1 if the field in the given row is `NULL`. It returns 0 if the field in the given row is `NOT NULL`. Field can be specified as column index (number) or fieldname (string). Row numbering starts at 0.

Note: This function used to be called `pg_fieldisnull()`.

pg_field_name (PHP 4 >= 4.2.0)

Returns the name of a field

```
string pg_field_name ( resource result, int field_number) \linebreak
```

pg_field_name() returns the name of the field occupying the given *field_number* in the given PostgreSQL *result* resource. Field numbering starts from 0.

Note: This function used to be called `pg_fieldname()`.

See also `pg_field_num()`.

pg_field_num (PHP 4 >= 4.2.0)

Returns the field number of the named field

```
int pg_field_num ( resource result, string field_name) \linebreak
```

pg_field_num() will return the number of the column (field) slot that corresponds to the *field_name* in the given PostgreSQL *result* resource. Field numbering starts at 0. This function will return -1 on error.

Note: This function used to be called `pg_fieldnum()`.

See also `pg_field_name()`.

pg_field_prtlen (PHP 4 >= 4.2.0)

Returns the printed length

`int pg_field_prtlen (resource result, int row_number, string field_name) \linebreak`

pg_field_prtlen() returns 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.

Note: This function used to be called `pg_field_prtlen()`.

See also `pg_field_size()`.

pg_field_size (PHP 4 >= 4.2.0)

Returns the internal storage size of the named field

`int pg_field_size (resource result, int field_number) \linebreak`

pg_field_size() returns 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.

Note: This function used to be called `pg_fieldsize()`.

See also `pg_field_len()` and `pg_field_type()`.

pg_field_type (PHP 4 >= 4.2.0)

Returns the type name for the corresponding field number

string **pg_field_type** (resource result, int field_number) \linebreak

pg_field_type() returns a string containing the type name of the given *field_number* in the given PostgreSQL *result* resource. Field numbering starts at 0.

Note: This function used to be called `pg_field_type()`.

See also **pg_field_len()** and **pg_field_name()**.

pg_free_result (PHP 4 >= 4.2.0)

Free result memory

bool **pg_free_result** (resource result) \linebreak

pg_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 **pg_free_result()** with the *result* resource as an argument and the associated result memory will be freed. It returns true on success and false if an error occurs.

Note: This function used to be called `pg_field_len()`.

See also **pg_query()**.

pg_get_result (PHP 4 >= 4.2.0)

Get asynchronous query result

resource **pg_get_result** ([resource connection]) \linebreak

pg_get_result() get result resource from async query executed by **pg_send_query()**. **pg_send_query()** can send multiple queries to PostgreSQL server and **pg_get_result()** is used to get query result one by one. It returns result resource. If there is no more results, it returns `FALSE`.

pg_host (PHP 3, PHP 4 >= 4.0.0)

Returns the host name associated with the connection

string **pg_host** (resource connection) \linebreak

pg_host() returns the host name of the given PostgreSQL *connection* resource is connected to.

See also `pg_connect()` and `pg_pconnect()`.

pg_insert (PHP 4 CVS only)

Insert array into table.

bool **pg_insert** (resource connection, string table_name, array assoc_array [, int options]) \linebreak

pg_insert() inserts `assoc_array` which has `field=>value` into table specified as `table_name`. If `options` is specified, `pg_convert()` is applied to `assoc_array` with specified option.

Example 1. pg_insert

```
<?php
$db = pg_connect ('dbname=foo');
// This is safe, since $_POST is converted autotmatically
$res = pg_insert($db, 'post_log', $_POST);
if ($res) {
    echo "POST data is succesfully logged\n";
}
else {
    echo "User must have sent wrong inputs\n";
}
?>
```

Note: This function is experimental.

See also `pg_convert()`

pg_last_error (PHP 4 >= 4.2.0)

Get the last error message string of a connection

string **pg_last_error** (resource connection) \linebreak

pg_last_error() returns the last error message for given *connection*.

Error messages may be overwritten by internal PostgreSQL(libpq) function calls. It may not return appropriate error message, if multiple errors are ocured inside a PostgreSQL module function.

Use `pg_result_error()`, `pg_result_status()` and `pg_connection_status()` for better error handling.

Note: This function used to be called `pg_errormessage()`.

See also `pg_result_error()`.

pg_last_notice (PHP 4 >= 4.0.6)

Returns the last notice message from PostgreSQL server

string **pg_last_notice** (resource connection) \linebreak

pg_last_notice() returns the last notice message from the PostgreSQL server specified by *connection*. The PostgreSQL server sends notice messages in several cases, e.g. if the transactions can't be continued. With **pg_last_notice()**, you can avoid issuing useless queries, by checking whether the notice is related to the transaction or not.

Warning

This function is EXPERIMENTAL and it is not fully implemented yet.

pg_last_notice() was added in PHP 4.0.6. However, PHP 4.0.6 has problem with notice message handling. Use of the PostgreSQL module with PHP 4.0.6 is not recommended even if you are not using **pg_last_notice()**.

This function is fully implemented in PHP 4.3.0. PHP earlier than PHP 4.3.0 ignores database connection parameter.

Notice message tracking can be set to optional by setting 1 for `pgsql.ignore_notice` ini from PHP 4.3.0.

Notice message logging can be set to optional by setting 0 for `pgsql.log_notice` ini from PHP 4.3.0. Unless `pgsql.ignore_notice` is set to 0, notice message cannot be logged.

See also `pg_query()` and `pg_last_error()`.

pg_last_oid (PHP 4 >= 4.2.0)

Returns the last object's oid

int **pg_last_oid** (resource result) \linebreak

pg_last_oid() is used to retrieve the `oid` assigned to an inserted tuple (record) if the result resource is used from the last command sent via `pg_query()` and was an SQL INSERT. Returns a positive integer if there was a valid `oid`. It returns `FALSE` if an error occurs or the last command sent via `pg_query()` was not an INSERT or INSERT is failed.

OID field became an optional field from PostgreSQL 7.2. When OID field is not defined in a table, programmer must use `pg_result_status()` to check if record is inserted successfully or not.

Note: This function used to be called `pg_getlastoid()`.

See also `pg_query()` and `pg_result_status()`

pg_lo_close (PHP 4 >= 4.2.0)

Close a large object

bool **pg_lo_close** (resource large_object) \linebreak

pg_lo_close() closes a Large Object. *large_object* is a resource for the large object from `pg_lo_open()`.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_loclose()`.

See also `pg_lo_open()`, `pg_lo_create()` and `pg_lo_import()`.

pg_lo_create (PHP 4 >= 4.2.0)

Create a large object

int **pg_lo_create** (resource connection) \linebreak

pg_lo_create() creates a Large Object and returns the `oid` of the large object. *connection* specifies a valid database connection opened by `pg_connect()` or `pg_pconnect()`. 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). It returns large object oid otherwise. It returns `FALSE`, if an error occurred,

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_locreate()`.

pg_lo_export (PHP 4 >= 4.2.0)

Export a large object to file

bool **pg_lo_export** (int oid, string pathname [, resource connection]) \linebreak

The *oid* argument specifies oid of the large object to export and the *pathname* argument specifies the pathname of the file. It returns `FALSE` if an error occurred, `TRUE` otherwise.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_loexport()`.

See also `pg_lo_import()`.

pg_lo_import (PHP 4 >= 4.2.0)

Import a large object from file

`int pg_lo_import (string pathname [, resource connection]) \linebreak`

The *pathname* argument specifies the pathname of the file to be imported as a large object. It returns `FALSE` if an error occurred, oid of the just created large object otherwise.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: When safe mode is enabled, PHP checks whether the file(s)/directories you are about to operate on have the same UID as the script that is being executed.

Note: This function used to be called `pg_loimport()`.

See also `pg_lo_export()` and `pg_lo_open()`.

pg_lo_open (PHP 4 >= 4.2.0)

Open a large object

`resource pg_lo_open (resource connection, int oid, string mode) \linebreak`

`pg_lo_open()` open a Large Object and returns large object resource. The resource encapsulates information about the connection. *oid* specifies a valid large object oid and *mode* can be either "r", "w", or "rw". It returns `FALSE` if there is an error.

Warning

Do not close the database connection before closing the large object resource.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_loopen()`.

See also `pg_lo_close()` and `pg_lo_create()`.

pg_lo_read (PHP 4 >= 4.2.0)

Read a large object

string **pg_lo_read** (resource `large_object`, int `len`) \linebreak

pg_lo_read() reads at most `len` bytes from a large object and returns it as a string. `large_object` specifies a valid large object resource and `len` specifies the maximum allowable size of the large object segment. It returns `FALSE` if there is an error.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_loread()`.

See also `pg_lo_read_all()`.

pg_lo_read_all (PHP 4 >= 4.2.0)

Read a entire large object and send straight to browser

int **pg_lo_read_all** (resource `large_object`) \linebreak

pg_lo_read_all() 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. It returns number of bytes read. It returns `FALSE`, if an error occurred.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_loreadall()`.

See also `pg_lo_read()`.

pg_lo_seek (PHP 4 >= 4.2.0)

Seeks position of large object

bool **pg_lo_seek** (resource large_object, int offset [, int whence]) \linebreak

pg_lo_seek() seeks position of large object resource. *whence* is PGSQL_SEEK_SET, PGSQL_SEEK_CUR or PGSQL_SEEK_END.

See also `pg_lo_tell()`.

pg_lo_tell (PHP 4 >= 4.2.0)

Returns current position of large object

int **pg_lo_tell** (resource large_object) \linebreak

pg_lo_tell() returns current position (offset from the beginning of large object).

See also `pg_lo_seek()`.

pg_lo_unlink (PHP 4 >= 4.2.0)

Delete a large object

bool **pg_lo_unlink** (resource connection, int oid) \linebreak

pg_lo_unlink() deletes a large object with the *oid*. It returns TRUE on success, otherwise returns FALSE.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_lo_unlink()`.

See also `pg_lo_create()` and `pg_lo_import()`.

pg_lo_write (PHP 4 >= 4.2.0)

Write a large object

int **pg_lo_write** (resource large_object, string data) \linebreak

pg_lo_write() writes at most to a large object from a variable *data* and returns the number of bytes actually written, or FALSE in the case of an error. *large_object* is a large object resource from `pg_lo_open()`.

To use the large object (lo) interface, it is necessary to enclose it within a transaction block.

Note: This function used to be called `pg_lo_write()`.

See also `pg_lo_create()` and `pg_lo_open()`.

pg_metadata (PHP 4 CVS only)

Get metadata for table.

array **pg_metadata** (resource connection, string table_name) \linebreak

pg_metadata() returns table definition for `table_name` as array. If there is error, it returns `FALSE`

Note: This function is experimental.

See also `pg_convert()`

pg_num_fields (PHP 4 >= 4.2.0)

Returns the number of fields

int **pg_num_fields** (resource result) \linebreak

pg_num_fields() returns the number of fields (columns) in a PostgreSQL *result*. The argument is a result resource returned by `pg_query()`. This function will return -1 on error.

Note: This function used to be called `pg_numfields()`.

See also `pg_num_rows()` and `pg_affected_rows()`.

pg_num_rows (PHP 4 >= 4.2.0)

Returns the number of rows

int **pg_num_rows** (resource result) \linebreak

pg_num_rows() will return the number of rows in a PostgreSQL *result* resource. *result* is a query result resource returned by `pg_query()`. This function will return -1 on error.

Note: Use `pg_affected_rows()` to get number of rows affected by INSERT, UPDATE and DELETE query.

Note: This function used to be called `pg_numrows()`.

See also `pg_num_fields()` and `pg_affected_rows()`.

pg_options (PHP 3, PHP 4 >= 4.0.0)

Get the options associated with the connection

string **pg_options** (resource connection) \linebreak

pg_options() will return a string containing the options specified on the given PostgreSQL *connection* resource.

pg_pconnect (PHP 3, PHP 4 >= 4.0.0)

Open a persistent PostgreSQL connection

int **pg_pconnect** (string connection_string) \linebreak

pg_pconnect() opens a connection to a PostgreSQL database. It returns a connection resource that is needed by other PostgreSQL functions.

For a description of the *connection_string* parameter, see `pg_connect()`.

To enable persistent connection, the `pgsql.allow_persistent` `php.ini` directive must be set to "On" (which is the default). The maximum number of persistent connection can be defined with the `pgsql.max_persistent` `php.ini` directive (defaults to -1 for no limit). The total number of connections can be set with the `pgsql.max_links` `php.ini` directive.

`pg_close()` will not close persistent links generated by **pg_pconnect()**.

See also `pg_connect()`.

pg_port (PHP 3, PHP 4 >= 4.0.0)

Return the port number associated with the connection

int **pg_port** (resource connection) \linebreak

pg_port() returns the port number that the given PostgreSQL *connection* resource 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, string data]) \linebreak

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** on success, **FALSE** on failure.

Note: The application must explicitly send the two characters "\." on the last 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
    $conn = pg_pconnect ("dbname=foo");
    pg_query($conn, "create table bar (a int4, b char(16), d float8)");
    pg_query($conn, "copy bar from stdin");
    pg_put_line($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_query (PHP 4 >= 4.2.0)

Execute a query

resource **pg_query** (resource connection, string query) \linebreak

pg_query() returns a query result resource if query could be executed. It returns **FALSE** on failure or if connection is not a valid connection. Details about the error can be retrieved using the `pg_last_error()` function if connection is valid. `pg_last_error()` sends an SQL statement to the PostgreSQL database specified by the *connection* resource. The *connection* must be a valid connection that was returned by `pg_connect()` or `pg_pconnect()`. The return value of this function is an query result resource to be used to access the results from other PostgreSQL functions such as `pg_fetch_array()`.

Note: *connection* is an optional parameter for **pg_query()**. If *connection* is not set, default connection is used. Default connection is the last connection made by `pg_connect()` or `pg_pconnect()`.

Although *connection* can be omitted, it is not recommended, since it could be a cause of hard to find bug in script.

Note: This function used to be called `pg_exec()`. `pg_exec()` is still available for compatibility reasons but users are encouraged to use the newer name.

See also `pg_connect()`, `pg_pconnect()`, `pg_fetch_array()`, `pg_fetch_object()`, `pg_num_rows()`, and `pg_affected_rows()`.

pg_result_error (PHP 4 >= 4.2.0)

Get error message associated with result

string **pg_result_error** (resource result) \linebreak

pg_result_error() returns error message associated with *result* resource. Therefore, user has better chance to get better error message than `pg_last_error()`.

See also `pg_query()`, `pg_send_query()`, `pg_get_result()`, `pg_last_error()` and `pg_last_notice()`

pg_result_status (PHP 4 >= 4.2.0)

Get status of query result

int **pg_result_status** (resource result) \linebreak

pg_result_status() returns status of result resource. Possible return values are `PGSQL_EMPTY_QUERY`, `PGSQL_COMMAND_OK`, `PGSQL_TUPLES_OK`, `PGSQL_COPY_TO`, `PGSQL_COPY_FROM`, `PGSQL_BAD_RESPONSE`, `PGSQL_NONFATAL_ERROR` and `PGSQL_FATAL_ERROR`.

See also `pg_connection_status()`.

pg_select (PHP 4 CVS only)

Select records.

array **pg_select** (resource connection, string table_name, array assoc_array [, int options]) \linebreak

pg_select() selects records specified by `assoc_array` which has `field=>value`. For successful query, it returns array contains all records and fields that match the condition specified by `assoc_array`. If `options` is specified, `pg_convert()` is applied to `assoc_array` with specified option.

Example 1. pg_select

```
<?php
    $db = pg_connect ('dbname=foo');
    // This is safe, since $_POST is converted automatically
    $rec = pg_select($db, 'post_log', $_POST);
    if ($rec) {
        echo "Records selected\n";
        var_dump($rec);
    }
    else {
        echo "User must have sent wrong inputs\n";
    }
?>
```

Note: This function is experimental.

See also `pg_convert()`

pg_send_query (PHP 4 >= 4.2.0)

Send asynchronous query

`bool pg_send_query (resource connection, string query) \linebreak` `bool pg_send_query (string query) \linebreak`

pg_send_query() send asynchronous query to the *connection*. Unlike `pg_query()`, it can send multiple query to PostgreSQL and get the result one by one using `pg_get_result()`. Script execution is not block while query is executing. Use `pg_connection_busy()` to check connection is busy (i.e. query is executing) Query may be canceled by calling `pg_cancel_query()`.

Although, user can send multiple query at once. User cannot send multiple query over busy connection. If query is sent while connection is busy, it waits until last query is finished and discards all result.

See also `pg_query()`, `pg_cancel_query()`, `pg_get_result()` and `pg_connection_busy()`

pg_set_client_encoding (PHP 3 CVS only, PHP 4 >= 4.0.3)

Set the client encoding

```
int pg_set_client_encoding ( [resource connection, string encoding] ) \linebreak
```

pg_set_client_encoding() sets 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. Available encoding depends on your PostgreSQL and libpq version. Refer to PostgreSQL manual for supported encodings for your PostgreSQL.

Note: This function requires PHP-4.0.3 or higher and PostgreSQL-7.0 or higher. Supported encoding depends on PostgreSQL version. Refer to PostgreSQL manual for details.

The function used to be called **pg_setclientencoding()**.

See also `pg_client_encoding()`.

pg_trace (PHP 4)

Enable tracing a PostgreSQL connection

```
bool pg_trace ( string pathname [, string mode [, resource connection]] ) \linebreak
```

pg_trace() enables tracing of the PostgreSQL frontend/backend communication to a debugging file specified as *pathname*. 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. For more information, refer to PostgreSQL manual.

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.

It returns `TRUE` if *pathname* could be opened for logging, `FALSE` otherwise.

See also `fopen()` and `pg_untrace()`.

pg_tty (PHP 3, PHP 4 >= 4.0.0)

Return the tty name associated with the connection

```
string pg_tty ( resource connection ) \linebreak
```

pg_tty() returns the tty name that server side debugging output is sent to on the given PostgreSQL *connection* resource.

pg_untrace (PHP 4)

Disable tracing of a PostgreSQL connection

bool **pg_untrace** ([resource connection]) \linebreak

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()`.

pg_update (PHP 4 CVS only)

Update table.

long **pg_update** (resource connection, string table_name, array condition, array data [, int options]) \linebreak

pg_update() updates records that matches *condition* with *data* If *options* is specified, `pg_convert()` is applied to *assoc_array* with specified options.

Example 1. pg_update

```

<?php
    $db = pg_connect ('dbname=foo');
    $data = array('field1'=>'AA', 'field2'=>'BB');
    // This is safe, since $_POST is converted automatically
    $res = pg_update($db, 'post_log', $_POST, $data);
    if ($res) {
        echo "Data is updated: $res\n";
    }
    else {
        echo "User must have sent wrong inputs\n";
    }
?>

```

Note: This function is experimental.

See also `pg_convert()`

LXXX. Process Control Functions

Process Control support in PHP implements the Unix style of process creation, program execution, signal handling and process termination. Process Control should not be enabled within a webserver environment and unexpected results may happen if any Process Control functions are used within a webserver environment.

This documentation is intended to explain the general usage of each of the Process Control functions. For detailed information about Unix process control you are encouraged to consult your systems documentation including `fork(2)`, `waitpid(2)` and `signal(2)` or a comprehensive reference such as *Advanced Programming in the UNIX Environment* by W. Richard Stevens (Addison-Wesley).

Process Control support in PHP is not enabled by default. You will need to use the `--enable-pcntl` configuration option when compiling PHP to enable Process Control support.

Note: Currently, this module will not function on non-Unix platforms (Windows).

The following list of signals are supported by the Process Control functions. Please see your systems `signal(7)` man page for details of the default behavior of these signals.

Table 1. Supported Signals

SIGFPE	SIGCONT	SIGKILL
SIGSTOP	SIGUSR1	SIGTSTP
SIGHUP	SIGUSR2	SIGTTIN
SIGINT	SIGSEGV	SIGTTOU
SIGQUIT	SIGPIPE	SIGURG
SIGILL	SIGALRM	SIGXCPU
SIGTRAP	SIGTERM	SIGXFSZ
SIGABRT	SIGSTKFLT	SIGVTALRM
SIGIOT	SIGCHLD	SIGPROF
SIGBUS	SIGCLD	SIGWINCH
SIGPOLL	SIGIO	SIGPWR
SIGSYS		

Process Control Example

This example forks off a daemon process with a signal handler.

Example 1. Process Control Example

```
<?php

$pid = pcntl_fork();
if ($pid == -1) {
    die("could not fork");
} else if ($pid) {
    exit(); // we are the parent
} else {
    // we are the child
}

// detach from the controlling terminal
if (!posix_setsid()) {
    die("could not detach from terminal");
}

// setup signal handlers
pcntl_signal(SIGTERM, "sig_handler");
pcntl_signal(SIGHUP, "sig_handler");

// loop forever performing tasks
while(1) {

    // do something interesting here

}

function sig_handler($signo) {

    switch($signo) {
        case SIGTERM:
            // handle shutdown tasks
            exit;
            break;
        case SIGHUP:
            // handle restart tasks
            break;
        default:
            // handle all other signals
    }
}

?>
```

pcntl_exec (PHP 4 >= 4.2.0)

Executes specified program in current process space

```
bool pcntl_exec ( string path [, array args [, array envs]]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

pcntl_fork (PHP 4 >= 4.1.0)

Forks the currently running process

```
int pcntl_fork ( void) \linebreak
```

The **pcntl_fork()** function creates a child process that differs from the parent process only in its PID and PPID. Please see your system's fork(2) man page for specific details as to how fork works on your system.

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and a PHP error is raised.

Example 1. pcntl_fork() Example

```
<?php

$pid = pcntl_fork();
if ($pid == -1) {
    die("could not fork");
} else if ($pid) {
    // we are the parent
} else {
    // we are the child
}

?>
```

See also `pcntl_waitpid()` and `pcntl_signal()`.

pcntl_signal (PHP 4 >= 4.1.0)

Installs a signal handler

```
bool pcntl_signal ( int signo, mixed handle) \linebreak
```

The **pcntl_signal()** function installs a new signal handler for the signal indicated by *signo*. The signal handler is set to *handler* which may be the name of a user created function, or either of the two global constants SIG_IGN or SIG_DFL.

pcntl_signal() returns TRUE on success or FALSE on failure.

Example 1. pcntl_signal() Example

```
<?php

// signal handler function
function sig_handler($signo) {

    switch($signo) {
        case SIGTERM:
            // handle shutdown tasks
            exit;
            break;
        case SIGHUP:
            // handle restart tasks
            break;
        case SIGUSR1:
            print "Caught SIGUSR1...\n";
            break;
        default:
            // handle all other signals
    }
}

print "Installing signal handler...\n";

// setup signal handlers
pcntl_signal(SIGTERM, "sig_handler");
pcntl_signal(SIGHUP, "sig_handler");
pcntl_signal(SIGUSR1, "sig_handler");

print "Generating signal SIGTERM to self...\n";

// send SIGUSR1 to current process id
posix_kill(posix_getpid(), SIGUSR1);

print "Done\n"

?>
```


See also `pcntl_fork()` and `pcntl_waitpid()`.

`pcntl_waitpid` (PHP 4 >= 4.1.0)

Waits on or returns the status of a forked child

```
int pcntl_waitpid ( int pid, int status, int options) \linebreak
```

The `pcntl_waitpid()` function suspends execution of the current process until a child as specified by the `pid` argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child as requested by `pid` has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed. Please see your system's `waitpid(2)` man page for specific details as to how `waitpid` works on your system.

`pcntl_waitpid()` returns the process ID of the child which exited, -1 on error or zero if `WNOHANG` was used and no child was available

The value of `pid` can be one of the following:

Table 1. possible values for `pid`

< -1	wait for any child process whose process group ID is equal to the absolute value of <code>pid</code> .
-1	wait for any child process; this is the same behaviour that the <code>wait</code> function exhibits.
0	wait for any child process whose process group ID is equal to that of the calling process.
> 0	wait for the child whose process ID is equal to the value of <code>pid</code> .

`pcntl_waitpid()` will store status information in the `status` parameter which can be evaluated using the following functions: `pcntl_wifexited()`, `pcntl_wifstopped()`, `pcntl_wifsignaled()`, `pcntl_wexitstatus()`, `pcntl_wtermsig()` and `pcntl_wstopsig()`.

The value of `options` is the value of zero or more of the following two global constants OR'ed together:

Table 2. possible values for `options`

<code>WNOHANG</code>	return immediately if no child has exited.
<code>WUNTRACED</code>	return for children which are stopped, and whose status has not been reported.

See also `pcntl_fork()`, `pcntl_signal()`, `pcntl_wifexited()`, `pcntl_wifstopped()`, `pcntl_wifsignaled()`, `pcntl_wexitstatus()`, `pcntl_wtermsig()` and `pcntl_wstopsig()`.

pcntl_wexitstatus (PHP 4 >= 4.1.0)

Returns the return code of a terminated child

int **pcntl_wexitstatus** (int status) \linebreak

Returns the return code of a terminated child. This function is only useful if `pcntl_wifexited()` returned `TRUE`.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()` and `pcntl_wifexited()`.

pcntl_wifexited (PHP 4 >= 4.1.0)

Returns `TRUE` if status code represents a successful exit

int **pcntl_wifexited** (int status) \linebreak

Returns `TRUE` if the child status code represents a successful exit.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()` and `pcntl_wexitstatus()`.

pcntl_wifsignaled (PHP 4 >= 4.1.0)

Returns `TRUE` if status code represents a termination due to a signal

int **pcntl_wifsignaled** (int status) \linebreak

Returns `TRUE` if the child process exited because of a signal which was not caught.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()` and `pcntl_signal()`.

pcntl_wifstopped (PHP 4 >= 4.1.0)

Returns `TRUE` if child process is currently stopped

int **pcntl_wifstopped** (int status) \linebreak

Returns `TRUE` if the child process which caused the return is currently stopped; this is only possible if the call to `pcntl_waitpid()` was done using the option `WUNTRACED`.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()`.

pcntl_wstopsig (PHP 4 >= 4.1.0)

Returns the signal which caused the child to stop

int **pcntl_wstopsig** (int status) \linebreak

Returns the number of the signal which caused the child to stop. This function is only useful if `pcntl_wifstopped()` returned `TRUE`.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()` and `pcntl_wifstopped()`.

pcntl_wtermsig (PHP 4 >= 4.1.0)

Returns the signal which caused the child to terminate

int **pcntl_wtermsig** (int status) \linebreak

Returns the number of the signal that caused the child process to terminate. This function is only useful if `pcntl_wifsignaled()` returned `TRUE`.

The parameter *status* is the status parameter supplied to a successful call to `pcntl_waitpid()`.

See also `pcntl_waitpid()`, `pcntl_signal()` and `pcntl_wifsignaled()`.

LXXXI. Program Execution functions

Those functions provides means to executes commands on the system itself, and means secure such commands. Those functions are also closely related to the backtick operator.

escapeshellarg (PHP 4 >= 4.0.3)

escape a string to be used as a shell argument

string **escapeshellarg** (string arg) \linebreak

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 >= 4.0.0)

escape shell metacharacters

string **escapeshellcmd** (string command) \linebreak

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 >= 4.0.0)

Execute an external program

string **exec** (string *command* [, string *array* [, int *return_var*]]) \linebreak

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.

Warning

If you are going to allow data coming from user input to be passed to this function, then you should be using `escapeshellarg()` or `escapeshellcmd()` to make sure that users cannot trick the system into executing arbitrary commands.

Note: 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 >= 4.0.0)

Execute an external program and display raw output

void **passthru** (string *command* [, int *return_var*]) \linebreak

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.

Warning

If you are going to allow data coming from user input to be passed to this function, then you should be using `escapeshellarg()` or `escapeshellcmd()` to make sure that users cannot trick the system into executing arbitrary commands.

Note: 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.

proc_close (PHP 4 CVS only)

Close a process opened by `proc_open` and return the exit code of that process.

`int proc_close (resource process) \linebreak`

proc_close() is similar to `popen()` except that it only works on processes opened by `proc_open()`. **proc_close()** waits for the process to terminate, and returns its exit code. If you have open pipes to that process, you should `fclose()` them prior to calling this function in order to avoid a deadlock - the child process may not be able to exit while the pipes are open.

proc_open (PHP 4 CVS only)

Execute a command and open file pointers for input/output

`resource proc_open (string cmd, array descriptorspec, array pipes) \linebreak`

proc_open() is similar to `popen()` but provides a much greater degree of control over the program execution. *cmd* is the command to be executed by the shell. *descriptorspec* is an indexed array where the key represents the descriptor number and the value represents how PHP will pass that descriptor to the child process. *pipes* will be set to an indexed array of file pointers that correspond to PHP's end of any pipes that are created. The return value is a resource representing the process; you should free it using `proc_close()` when you are finished with it.

```
$descriptorspec = array(
    0 => array("pipe", "r"), // stdin is a pipe that the child will read from
    1 => array("pipe", "w"), // stdout is a pipe that the child will write to
    2 => array("file", "/tmp/error-output.txt", "a"), // stderr is a file to write to
);
$process = proc_open("php", $descriptorspec, $pipes);
if (is_resource($process)) {
    // $pipes now looks like this:
    // 0 => writable handle connected to child stdin
    // 1 => readable handle connected to child stdout
    // Any error output will be appended to /tmp/error-output.txt

    fwrite($pipes[0], "<?php echo \"Hello World!\"; ?>");
    fclose($pipes[0]);
}
```

```

while(!feof($pipes[1])) {
    echo fgets($pipes[1], 1024);
}
fclose($pipes[1]);
// It is important that you close any pipes before calling
// proc_close in order to avoid a deadlock
$return_value = proc_close($process);

echo "command returned $return_value\n";
}

```

The file descriptor numbers in *descriptorspec* are not limited to 0, 1 and 2 - you may specify any valid file descriptor number and it will be passed to the child process. This allows your script to interoperate with other scripts that run as "co-processes". In particular, this is useful for passing passphrases to programs like PGP, GPG and openssl in a more secure manner. It is also useful for reading status information provided by those programs on auxiliary file descriptors.

Note: Windows compatibility: Descriptors beyond 2 (stderr) are made available to the child process as inheritable handles, but since the Windows architecture does not associate file descriptor numbers with low-level handles, the child process does not (yet) have a means of accessing those handles. Stdin, stdout and stderr work as expected.

Note: This function was introduced in PHP 4.3.0.

Note: If you only need a uni-directional (one-way) process pipe, use `popen()` instead, as it is much easier to use.

See also `exec()`, `system()`, `passthru()`, `popen()`, `escapeshellcmd()`, and the backtick operator.

shell_exec (PHP 4 >= 4.0.0)

Execute command via shell and return complete output as string

string **shell_exec** (string cmd) \linebreak

This function is identical to the backtick operator.

system (PHP 3, PHP 4 >= 4.0.0)

Execute an external program and display output

string **system** (string *command* [, int *return_var*]) \linebreak

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.

Warning

If you are going to allow data coming from user input to be passed to this function, then you should be using `escapeshellarg()` or `escapeshellcmd()` to make sure that users cannot trick the system into executing arbitrary commands.

Note: 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.

LXXXII. Printer functions

These functions are only available under Windows 9.x, ME, NT4 and 2000. They have been added in PHP 4 (4.0.4).

printer_abort (unknown)

Deletes the printer's spool file

```
void printer_abort ( resource handle) \linebreak
```

This function deletes the printers spool file.

handle must be a valid handle to a printer.

Example 1. printer_abort() example

```
$handle = printer_open();
printer_abort($handle);
printer_close($handle);
```

printer_close (unknown)

Close an open printer connection

```
void printer_close ( resource handle) \linebreak
```

This function closes the printer connection. **printer_close()** also closes the active device context.

handle must be a valid handle to a printer.

Example 1. printer_close() example

```
$handle = printer_open();
printer_close($handle);
```

printer_create_brush (unknown)

Create a new brush

```
mixed printer_create_brush ( int style, string color) \linebreak
```

The function creates a new brush and returns a handle to it. A brush is used to fill shapes. For an example see `printer_select_brush()`. *color* must be a color in RGB hex format, i.e. "000000" for black, *style* must be one of the following constants:

- `PRINTER_BRUSH_SOLID`: creates a brush with a solid color.
- `PRINTER_BRUSH_DIAGONAL`: creates a brush with a 45-degree upward left-to-right hatch (/).
- `PRINTER_BRUSH_CROSS`: creates a brush with a cross hatch (+).
- `PRINTER_BRUSH_DIAGCROSS`: creates a brush with a 45 cross hatch (x).
- `PRINTER_BRUSH_FDIAGONAL`: creates a brush with a 45-degree downward left-to-right hatch (\).
- `PRINTER_BRUSH_HORIZONTAL`: creates a brush with a horizontal hatch (-).
- `PRINTER_BRUSH_VERTICAL`: creates a brush with a vertical hatch (|).
- `PRINTER_BRUSH_CUSTOM`: creates a custom brush from an BMP file. The second parameter is used to specify the BMP instead of the RGB color code.

printer_create_dc (unknown)

Create a new device context

void **printer_create_dc** (resource handle) \linebreak

The function creates a new device context. A device context is used to customize the graphic objects of the document. *handle* must be a valid handle to a printer.

Example 1. printer_create_dc() example

```
$handle = printer_open();
printer_start_doc($handle);
printer_start_page($handle);

printer_create_dc($handle);
/* do some stuff with the dc */
printer_set_option($handle, PRINTER_TEXT_COLOR, "333333");
printer_draw_text($handle, 1, 1, "text");
printer_delete_dc($handle);

/* create another dc */
printer_create_dc($handle);
printer_set_option($handle, PRINTER_TEXT_COLOR, "000000");
printer_draw_text($handle, 1, 1, "text");
/* do some stuff with the dc */

printer_delete_dc($handle);

printer_endpage($handle);
printer_end_doc($handle);
printer_close($handle);
```

printer_create_font (unknown)

Create a new font

mixed **printer_create_font** (string face, int height, int width, int font_weight, bool italic, bool underline, bool
strikeout, int orientaton) \linebreak

The function creates a new font and returns a handle to it. A font is used to draw text. For an example see `printer_select_font()`. *face* must be a string specifying the font face. *height* specifies the font height, and *width* the font width. The *font_weight* specifies the font weight (400 is normal), and can be one of the following predefined constants.

- *PRINTER_FW_THIN*: sets the font weight to thin (100).
- *PRINTER_FW_ULTRALIGHT*: sets the font weight to ultra light (200).
- *PRINTER_FW_LIGHT*: sets the font weight to light (300).
- *PRINTER_FW_NORMAL*: sets the font weight to normal (400).
- *PRINTER_FW_MEDIUM*: sets the font weight to medium (500).
- *PRINTER_FW_BOLD*: sets the font weight to bold (700).
- *PRINTER_FW_ULTRABOLD*: sets the font weight to ultra bold (800).
- *PRINTER_FW_HEAVY*: sets the font weight to heavy (900).

italic can be TRUE or FALSE, and sets whether the font should be italic.

underline can be TRUE or FALSE, and sets whether the font should be underlined.

strikeout can be TRUE or FALSE, and sets whether the font should be striked out.

orientation specifies a rotation. For an example see `printer_select_font()`.

printer_create_pen (unknown)

Create a new pen

mixed **printer_create_pen** (int style, int width, string color) \linebreak

The function creates a new pen and returns a handle to it. A pen is used to draw lines and curves. For an example see `printer_select_pen()`. *color* must be a color in RGB hex format, i.e. "000000" for black, *width* specifies the width of the pen whereas *style* must be one of the following constants:

- *PRINTER_PEN_SOLID*: creates a solid pen.
- *PRINTER_PEN_DASH*: creates a dashed pen.
- *PRINTER_PEN_DOT*: creates a dotted pen.
- *PRINTER_PEN_DASHDOT*: creates a pen with dashes and dots.
- *PRINTER_PEN_DASHDOTDOT*: creates a pen with dashes and double dots.

- `PRINTER_PEN_INVISIBLE`: creates an invisible pen.

printer_delete_brush (unknown)

Delete a brush

bool **printer_delete_brush** (resource handle) \linebreak

The function deletes the selected brush. For an example see `printer_select_brush()`. It returns `TRUE` on success, or `FALSE` otherwise. *handle* must be a valid handle to a brush.

printer_delete_dc (unknown)

Delete a device context

bool **printer_delete_dc** (resource handle) \linebreak

The function deletes the device context and returns `TRUE` on success, or `FALSE` if an error occurred. For an example see `printer_create_dc()`. *handle* must be a valid handle to a printer.

printer_delete_font (unknown)

Delete a font

bool **printer_delete_font** (resource handle) \linebreak

The function deletes the selected font. For an example see `printer_select_font()`. It returns `TRUE` on success, or `FALSE` otherwise. *handle* must be a valid handle to a font.

printer_delete_pen (unknown)

Delete a pen

bool **printer_delete_pen** (resource handle) \linebreak

The function deletes the selected pen. For an example see `printer_select_pen()`. It returns `TRUE` on success, or `FALSE` otherwise. *handle* must be a valid handle to a pen.

printer_draw_bmp (unknown)

Draw a bmp

void **printer_draw_bmp** (resource handle, string filename, int x, int y) \linebreak

The function simply draws an bmp the bitmap *filename* at position *x, y*. *handle* must be a valid handle to a printer.

The function returns TRUE on success, or otherwise FALSE.

Example 1. printer_draw_bmp() example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

printer_draw_bmp($handle, "c:\\image.bmp", 1, 1);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);
```

printer_draw_chord (unknown)

Draw a chord

void **printer_draw_chord** (resource handle, int rec_x, int rec_y, int rec_x1, int rec_y1, int rad_x, int rad_y, int rad_x1, int rad_y1) \linebreak

The function simply draws an chord. *handle* must be a valid handle to a printer.

rec_x is the upper left x coordinate of the bounding rectangle.

rec_y is the upper left y coordinate of the bounding rectangle.

rec_x1 is the lower right x coordinate of the bounding rectangle.

rec_y1 is the lower right y coordinate of the bounding rectangle.

rad_x is x coordinate of the radial defining the beginning of the chord.

rad_y is y coordinate of the radial defining the beginning of the chord.

rad_x1 is x coordinate of the radial defining the end of the chord.

rad_y1 is y coordinate of the radial defining the end of the chord.

Example 1. printer_draw_chord() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "2222FF");
printer_select_brush($handle, $brush);

printer_draw_chord($handle, 1, 1, 500, 500, 1, 1, 500, 1);

printer_delete_brush($brush);
printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_draw_ellipse (unknown)

Draw an ellipse

void **printer_draw_ellipse** (resource handle, int ul_x, int ul_y, int lr_x, int lr_y) \linebreak

The function simply draws an ellipse. *handle* must be a valid handle to a printer.

ul_x is the upper left x coordinate of the ellipse.

ul_y is the upper left y coordinate of the ellipse.

lr_x is the lower right x coordinate of the ellipse.

lr_y is the lower right y coordinate of the ellipse.

Example 1. printer_draw_ellipse() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "2222FF");
printer_select_brush($handle, $brush);

```



```

printer_draw_ellipse($handle, 1, 1, 500, 500);

printer_delete_brush($brush);
printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_draw_line (unknown)

Draw a line

```
void printer_draw_line ( resource printer_handle, int from_x, int from_y, int to_x, int to_y) \linebreak
```

The function simply draws a line from position *from_x*, *from_y* to position *to_x*, *to_y* using the selected pen. *printer_handle* must be a valid handle to a printer.

Example 1. printer_draw_line() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 30, "000000");
printer_select_pen($handle, $pen);

printer_draw_line($handle, 1, 10, 1000, 10);
printer_draw_line($handle, 1, 60, 500, 60);

printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_draw_pie (unknown)

Draw a pie

```
void printer_draw_pie ( resource handle, int rec_x, int rec_y, int rec_x1, int rec_y1, int rad1_x, int rad1_y, int rad2_x, int rad2_y) \linebreak
```

The function simply draws an pie. *handle* must be a valid handle to a printer.

rec_x is the upper left x coordinate of the bounding rectangle.

rec_y is the upper left y coordinate of the bounding rectangle.

rec_x1 is the lower right x coordinate of the bounding rectangle.

rec_y1 is the lower right y coordinate of the bounding rectangle.

rad1_x is x coordinate of the first radial's ending.

rad1_y is y coordinate of the first radial's ending.

rad2_x is x coordinate of the second radial's ending.

rad2_y is y coordinate of the second radial's ending.

Example 1. **printer_draw_chord()** example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "2222FF");
printer_select_brush($handle, $brush);

printer_draw_pie($handle, 1, 1, 500, 500, 1, 1, 500, 1);

printer_delete_brush($brush);
printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);
```

printer_draw_rectangle (unknown)

Draw a rectangle

```
void printer_draw_rectangle ( resource handle, int ul_x, int ul_y, int lr_x, int lr_y) \linebreak
```

The function simply draws a rectangle.

handle must be a valid handle to a printer.

ul_x is the upper left x coordinate of the rectangle.

ul_y is the upper left y coordinate of the rectangle.

lr_x is the lower right x coordinate of the rectangle.

lr_y is the lower right y coordinate of the rectangle.

Example 1. `printer_draw_rectangle()` example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "2222FF");
printer_select_brush($handle, $brush);

printer_draw_rectangle($handle, 1, 1, 500, 500);

printer_delete_brush($brush);
printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);
```

`printer_draw_roundrect` (unknown)

Draw a rectangle with rounded corners

void **printer_draw_roundrect** (resource handle, int *ul_x*, int *ul_y*, int *lr_x*, int *lr_y*, int *width*, int *height*) \line-break

The function simply draws a rectangle with rounded corners.

handle must be a valid handle to a printer.

ul_x is the upper left x coordinate of the rectangle.

ul_y is the upper left y coordinate of the rectangle.

lr_x is the lower right x coordinate of the rectangle.

lr_y is the lower right y coordinate of the rectangle.

width is the width of the ellipse.

height is the height of the ellipse.

Example 1. printer_draw_roundrect() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "2222FF");
printer_select_brush($handle, $brush);

printer_draw_roundrect($handle, 1, 1, 500, 500, 200, 200);

printer_delete_brush($brush);
printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_draw_text (unknown)

Draw text

void **printer_draw_text** (resource printer_handle, string text, int x, int y) \linebreak

The function simply draws *text* at position *x*, *y* using the selected font. *printer_handle* must be a valid handle to a printer.

Example 1. printer_draw_text() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$font = printer_create_font("Arial", 72, 48, 400, false, false, false, 0);
printer_select_font($handle, $font);
printer_draw_text($handle, "test", 10, 10);
printer_delete_font($font);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_end_doc (unknown)

Close document

bool **printer_end_doc** (resource handle) \linebreak

Closes a new document in the printer spooler. The document is now ready for printing. For an example see `printer_start_doc()`. *handle* must be a valid handle to a printer.

printer_end_page (unknown)

Close active page

bool **printer_end_page** (resource handle) \linebreak

The function closes the active page in the active document. For an example see `printer_start_doc()`. *handle* must be a valid handle to a printer.

printer_get_option (unknown)

Retrieve printer configuration data

mixed **printer_get_option** (resource handle, string option) \linebreak

The function retrieves the configuration setting of *option*. *handle* must be a valid handle to a printer. Take a look at `printer_set_option()` for the settings that can be retrieved, additionally the following settings can be retrieved:

- `PRINTER_DEVICENAME` returns the devicename of the printer.
- `PRINTER_DRIVERVERSION` returns the printer driver version.

Example 1. printer_get_option() example

```
$handle = printer_open();
print printer_get_option($handle, PRINTER_DRIVERVERSION);
printer_close($handle);
```

printer_list (unknown)

Return an array of printers attached to the server

array **printer_list** (int enumtype [, string name [, int level]]) \linebreak

The function enumerates available printers and their capabilities. *level* sets the level of information request. Can be 1,2,4 or 5. *enumtype* must be one of the following predefined constants:

- *PRINTER_ENUM_LOCAL*: enumerates the locally installed printers.
- *PRINTER_ENUM_NAME*: enumerates the printer of *name*, can be a server, domain or print provider.
- *PRINTER_ENUM_SHARED*: this parameter can't be used alone, it has to be OR'ed with other parameters, i.e. *PRINTER_ENUM_LOCAL* to detect the locally shared printers.
- *PRINTER_ENUM_DEFAULT*: (Win9.x only) enumerates the default printer.
- *PRINTER_ENUM_CONNECTIONS*: (WinNT/2000 only) enumerates the printers to which the user has made connections.
- *PRINTER_ENUM_NETWORK*: (WinNT/2000 only) enumerates network printers in the computer's domain. Only valid if *level* is 1.
- *PRINTER_ENUM_REMOTE*: (WinNT/2000 only) enumerates network printers and print servers in the computer's domain. Only valid if *level* is 1.

Example 1. printer_list() example

```
/* detect locally shared printer */
var_dump( printer_list(PRINTER_ENUM_LOCAL | PRINTER_ENUM_SHARED) );
```

printer_logical_fontheight (unknown)

Get logical font height

int **printer_logical_fontheight** (resource handle, int height) \linebreak

The function calculates the logical font height of *height*. *handle* must be a valid handle to a printer.

Example 1. printer_logical_fontheight() example

```
$handle = printer_open();
print printer_logical_fontheight($handle, 72);
printer_close($handle);
```

printer_open (unknown)

Open connection to a printer

mixed **printer_open** ([string devicename]) \linebreak

This function tries to open a connection to the printer *devicename*, and returns a handle on success or `FALSE` on failure.

If no parameter was given it tries to open a connection to the default printer (if not specified in `php.ini` as `printer.default_printer`, `php` tries to detect it).

printer_open() also starts a device context.

Example 1. printer_open() example

```
$handle = printer_open("HP Deskjet 930c");
$handle = printer_open();
```

printer_select_brush (unknown)

Select a brush

void **printer_select_brush** (resource printer_handle, resource brush_handle) \linebreak

The function selects a brush as the active drawing object of the actual device context. A brush is used to fill shapes. If you draw an rectangle the brush is used to draw the shapes, while the pen is used to draw the border. If you haven't selected a brush before drawing shapes, the shape won't be filled.

printer_handle must be a valid handle to a printer. *brush_handle* must be a valid handle to a brush.

Example 1. printer_select_brush() example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 2, "000000");
printer_select_pen($handle, $pen);
$brush = printer_create_brush(PRINTER_BRUSH_CUSTOM, "c:\\brush.bmp");
printer_select_brush($handle, $brush);
```

```

printer_draw_rectangle($handle, 1,1,500,500);

printer_delete_brush($brush);

$brush = printer_create_brush(PRINTER_BRUSH_SOLID, "000000");
printer_select_brush($handle, $brush);
printer_draw_rectangle($handle, 1,501,500,1001);
printer_delete_brush($brush);

printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```

printer_select_font (unknown)

Select a font

void **printer_select_font** (resource printer_handle, resource font_handle) \linebreak

The function selects a font to draw text. *printer_handle* must be a valid handle to a printer. *font_handle* must be a valid handle to a font.

Example 1. printer_select_font() example

```

$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$font = printer_create_font("Arial", 148, 76, PRINTER_FW_MEDIUM, false, false, false, -
50);
printer_select_font($handle, $font);
printer_draw_text($handle, "PHP is simply cool", 40, 40);
printer_delete_font($font);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);

```


printer_select_pen (unknown)

Select a pen

```
void printer_select_pen ( resource printer_handle, resource pen_handle) \linebreak
```

The function selects a pen as the active drawing object of the actual device context. A pen is used to draw lines and curves. I.e. if you draw a single line the pen is used. If you draw an rectangle the pen is used to draw the borders, while the brush is used to fill the shape. If you haven't selected a pen before drawing shapes, the shape won't be outlined. *printer_handle* must be a valid handle to a printer. *pen_handle* must be a valid handle to a pen.

Example 1. printer_select_pen() example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

$pen = printer_create_pen(PRINTER_PEN_SOLID, 30, "2222FF");
printer_select_pen($handle, $pen);

printer_draw_line($handle, 1, 60, 500, 60);

printer_delete_pen($pen);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);
```

printer_set_option (unknown)

Configure the printer connection

```
bool printer_set_option ( resource handle, int option, mixed value) \linebreak
```

The function sets the following options for the current connection: *handle* must be a valid handle to a printer. For *option* can be one of the following constants:

- *PRINTER_COPIES*: sets how many copies should be printed, *value* must be an integer.
- *PRINTER_MODE*: specifies the type of data (text, raw or emf), *value* must be a string.
- *PRINTER_TITLE*: specifies the name of the document, *value* must be a string.
- *PRINTER_ORIENTATION*: specifies the orientation of the paper, *value* can be either *PRINTER_ORIENTATION_PORTRAIT* or *PRINTER_ORIENTATION_LANDSCAPE*
- *PRINTER_RESOLUTION_Y*: specifies the y-resolution in DPI, *value* must be an integer.

- *PRINTER_RESOLUTION_X*: specifies the x-resolution in DPI, *value* must be an integer.
- *PRINTER_PAPER_FORMAT*: specifies the a predefined paper format, set *value* to *PRINTER_FORMAT_CUSTOM* if you want to specify a custom format with *PRINTER_PAPER_WIDTH* and *PRINTER_PAPER_LENGTH*. *value* can be one of the following constants.
 - *PRINTER_FORMAT_CUSTOM*: let's you specify a custom paper format.
 - *PRINTER_FORMAT_LETTER*: specifies standard letter format (8 1/2- by 11-inches).
 - *PRINTER_FORMAT_LETTER*: specifies standard legal format (8 1/2- by 14-inches).
 - *PRINTER_FORMAT_A3*: specifies standard A3 format (297- by 420-millimeters).
 - *PRINTER_FORMAT_A4*: specifies standard A4 format (210- by 297-millimeters).
 - *PRINTER_FORMAT_A5*: specifies standard A5 format (148- by 210-millimeters).
 - *PRINTER_FORMAT_B4*: specifies standard B4 format (250- by 354-millimeters).
 - *PRINTER_FORMAT_B5*: specifies standard B5 format (182- by 257-millimeter).
 - *PRINTER_FORMAT_FOLIO*: specifies standard FOLIO format (8 1/2- by 13-inch).
- *PRINTER_PAPER_LENGTH*: if *PRINTER_PAPER_FORMAT* is set to *PRINTER_FORMAT_CUSTOM*, *PRINTER_PAPER_LENGTH* specifies a custom paper length in mm, *value* must be an integer.
- *PRINTER_PAPER_WIDTH*: if *PRINTER_PAPER_FORMAT* is set to *PRINTER_FORMAT_CUSTOM*, *PRINTER_PAPER_WIDTH* specifies a custom paper width in mm, *value* must be an integer.
- *PRINTER_SCALE*: specifies the factor by which the printed output is to be scaled. the page size is scaled from the physical page size by a factor of *scale*/100. for example if you set the scale to 50, the output would be half of it's original size. *value* must be an integer.
- *PRINTER_BACKGROUND_COLOR*: specifies the background color for the actual device context, *value* must be a string containing the rgb information in hex format i.e. "005533".
- *PRINTER_TEXT_COLOR*: specifies the text color for the actual device context, *value* must be a string containing the rgb information in hex format i.e. "005533".
- *PRINTER_TEXT_ALIGN*: specifies the text alignment for the actual device context, *value* can be combined through OR'ing the following constants:
 - *PRINTER_TA_BASELINE*: text will be aligned at the base line.
 - *PRINTER_TA_BOTTOM*: text will be aligned at the bottom.
 - *PRINTER_TA_TOP*: text will be aligned at the top.
 - *PRINTER_TA_CENTER*: text will be aligned at the center.
 - *PRINTER_TA_LEFT*: text will be aligned at the left.
 - *PRINTER_TA_RIGHT*: text will be aligned at the right.

Example 1. printer_set_option() example

```
$handle = printer_open();
printer_set_option($handle, PRINTER_SCALE, 75);
printer_set_option($handle, PRINTER_TEXT_ALIGN, PRINTER_TA_LEFT);
printer_close($handle);
```

printer_start_doc (unknown)

Start a new document

```
bool printer_start_doc ( resource handle [, string document]) \linebreak
```

The function creates a new document in the printer spooler. A document can contain multiple pages, it's used to schedule the print job in the spooler. *handle* must be a valid handle to a printer. The optional parameter *document* can be used to set an alternative document name.

Example 1. printer_start_doc() example

```
$handle = printer_open();
printer_start_doc($handle, "My Document");
printer_start_page($handle);

printer_end_page($handle);
printer_end_doc($handle);
printer_close($handle);
```

printer_start_page (unknown)

Start a new page

```
bool printer_start_page ( resource handle) \linebreak
```

The function creates a new page in the active document. For an example see `printer_start_doc()`. *handle* must be a valid handle to a printer.

printer_write (unknown)

Write data to the printer

bool **printer_write** (resource handle, string content) \linebreak

Writes *content* directly to the printer, and returns TRUE on success or FALSE if it failed.

handle must be a valid handle to a printer.

Example 1. printer_write() example

```
$handle = printer_open();  
printer_write($handle, "Text to print");  
printer_close($handle);
```

LXXXIII. 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) \linebreak
```

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) \linebreak
```

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

```
bool pspell_check ( int dictionary_link, string word) \linebreak
```

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) \linebreak

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]]]) \linebreak

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, "en");
```

pspell_config_ignore (PHP 4 >= 4.0.2)

Ignore words less than N characters long

```
int pspell_config_ignore ( int dictionary_link, int n) \linebreak
```

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 than 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) \linebreak
```

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) \linebreak
```

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) \linebreak
```

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_runtogether (PHP 4 >= 4.0.2)

Consider run-together words as valid compounds

```
int pspell_config_runtogether ( int dictionary_link, bool flag) \linebreak
```

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, bool flag) \linebreak
```

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]]]]) \linebreak
```

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 ) \linebreak
```

pspell_new_config() opens up a new dictionary with settings specified in a config, created with 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");
$pspell_link = pspell_new_config ($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]]]]) \linebreak
```

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) \linebreak
```

`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) \linebreak

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) \linebreak

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");

    foreach ($suggestions as $suggestion) {
        echo "Possible spelling: $suggestion<br>";
    }
}
```


LXXXIV. 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.

Requirements

To use the readline functions, you need to install libreadline and compile PHP with support for readline.

Installation

To compile PHP with readline support, you need to configure PHP `--with-readline` after you've installed libreadline. You can find libreadline on the home page of the GNU Readline project, at <http://cnswww.cns.cwru.edu/~chet/readline/rltop.html>. It's maintained by Chet Ramey, who's also the author of Bash.

Runtime Configuration

This extension does not define any configuration directives.

Resource types

This extension does not define any resource types.

Predefined constants

This extension does not define any constants.

readline (PHP 4 >= 4.0.0)

Reads a line

string **readline** ([string prompt]) \linebreak

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 >= 4.0.0)

Adds a line to the history

void **readline_add_history** (string line) \linebreak

This function adds a line to the command line history.

readline_clear_history (PHP 4 >= 4.0.0)

Clears the history

bool **readline_clear_history** (void) \linebreak

This function clears the entire command line history.

readline_completion_function (PHP 4 >= 4.0.0)

Registers a completion function

bool **readline_completion_function** (string line) \linebreak

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 >= 4.0.0)

Gets/sets various internal readline variables

mixed **readline_info** ([string varname [, string newvalue]]) \linebreak

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 >= 4.0.0)

Lists the history

array **readline_list_history** (void) \linebreak

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 >= 4.0.0)

Reads the history

bool **readline_read_history** (string filename) \linebreak

This function reads a command history from a file.

readline_write_history (PHP 4 >= 4.0.0)

Writes the history

bool readline_write_history (string filename) \linebreak

This function writes the command history to a file.

LXXXV. 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 (PHP 4 >= 4.0.0)

Recode a string according to a recode request

string **recode** (string request, string string) \linebreak

Note: This is an alias for `recode_string()`. It has been added in PHP 4.

recode_file (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Recode from file to file according to recode request

bool **recode_file** (string request, resource input, resource output) \linebreak

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);
```

recode_string (PHP 3>= 3.0.13, PHP 4 >= 4.0.0)

Recode a string according to a recode request

string **recode_string** (string request, string string) \linebreak

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: &acute;")
```

LXXXVI. 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.

PHP also supports regular expressions using a POSIX-extended syntax using the POSIX-extended regex functions..

Requirements

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. It is available at <ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/>.

Installation

Beginning with PHP 4.2.0 this function are enabled by default. For older versions you have to configure and compile PHP with `--with-pcre-regex[=DIR]` in order to use these functions. You can disable

the pcre functions with `--without-pcre-regex`.

Runtime Configuration

This extension does not define any configuration directives.

Resource types

This extension does not define any resource types.

Predefined constants

`PREG_PATTERN_ORDER` `PREG_SET_ORDER` `PREG_SPLIT_NO_EMPTY` `PREG_SPLIT_DELIM_CAPTURE`

Examples

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

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 *D* 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.

Only preg_replace() uses this modifier; it is ignored by other PCRE functions.

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.

u (PCRE_UTF8)

This modifier turns on additional functionality of PCRE that is incompatible with Perl. Pattern strings are treated as UTF-8. This modifier is available from PHP 4.1.0 or greater.

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.

Introduction

The syntax and semantics of the regular expressions supported 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.

Meta-characters

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 end 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
 }
 end 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-alphameric 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-alphameric 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:

<code>\a</code>	alarm, that is, the BEL character (hex 07)
<code>\cx</code>	"control-x", where x is any character
<code>\e</code>	escape (hex 1B)
<code>\f</code>	formfeed (hex 0C)
<code>\n</code>	newline (hex 0A)
<code>\r</code>	carriage return (hex 0D)
<code>\t</code>	tab (hex 09)
<code>\xhh</code>	character with hex code hh
<code>\ddd</code>	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

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 `[][\^_‘xyzabc]`, 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 res-

stricted set of characters than the matching lower case type. For example, the class `[^\W_]` matches any letter or digit, but not underscore.

All non-alphameric 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

```
(?i: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

```
^.*?\s*/
```

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 caselessly.

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. Back-

tracking 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 subpat-

tern 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

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

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

```
(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)
```

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.

Performances

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.

preg_grep (PHP 4 >= 4.0.0)

Return array entries that match the pattern

array **preg_grep** (string pattern, array input) \linebreak

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
$fl_array = preg_grep ("/^(\\d+)?\\.\\d+$/", $array);
```

preg_match (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Perform a regular expression match

int **preg_match** (string pattern, string subject [, array matches]) \linebreak

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. `$matches[0]` will contain the text that matched the full pattern, `$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:\\\\\/)?([^\\/]+)/i",
"http://www.php.net/index.html", $matches);
$host = $matches[2];
// get last two segments of host name
preg_match ("/[^\.\\/]+\.[^\.\\/]+$/", $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 >= 4.0.0)

Perform a global regular expression match

int **preg_match_all** (string *pattern*, string *subject*, array *matches* [, int *order*]) \linebreak

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 ("/\((? (\d{3})? \)? (?1) [\-\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 second set of parentheses in the regular expression
// itself, which would be the ([\w]+) in this case. The extra backslash is
// required because the string is in double quotes.
$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_quote (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Quote regular expression characters

string **preg_quote** (string *str* [, string *delimiter*]) \linebreak

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_replace (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Perform a regular expression search and replace

mixed **preg_replace** (mixed pattern, mixed replacement, mixed subject [, int limit]) \linebreak

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 matches are found, the new *subject* will be returned, otherwise *subject* 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()**.

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_replace_callback (PHP 4 >= 4.0.5)

Perform a regular expression search and replace using a callback

mixed **preg_replace_callback** (mixed pattern, mixed callback, mixed subject [, int limit]) \linebreak

The behavior of this function is almost identical to `preg_replace()`, except for the fact that instead of *replacement* parameter, one should specify a *callback* that will be called and passed an array of matched elements in the subject string. The callback should return the replacement string. This function was added in PHP 4.0.5.

See also `preg_replace()`.

preg_split (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Split string by a regular expression

array **preg_split** (string pattern, string subject [, int limit [, int flags]]) \linebreak

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, and if *limit* is -1, it actually means "no limit", which is useful for specifying the *flags*.

flags can be any combination of the following flags (combined with bitwise | operator):

PREG_SPLIT_NO_EMPTY

If this flag is set, only non-empty pieces will be returned by `preg_split()`.

PREG_SPLIT_DELIM_CAPTURE

If this flag is set, parenthesized expression in the delimiter pattern will be captured and returned as well. This flag was added for 4.0.5.

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");
```

Example 2. Splitting a string into component characters.

```
$str = 'string';  
$chars = preg_split('///', $str, -1, PREG_SPLIT_NO_EMPTY);  
print_r($chars);
```

See also `spliti()`, `split()`, `implode()`, `preg_match()`, `preg_match_all()`, and `preg_replace()`.

LXXXVII. qtdom functions

qdom_error (PHP 4 >= 4.0.5)

Returns the error string from the last QDOM operation or FALSE if no errors occurred

string **qdom_error** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

qdom_tree (PHP 4 >= 4.0.4)

creates a tree of an xml string

object **qdom_tree** (string) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

LXXXVIII. Regular Expression Functions (POSIX Extended)

Note: PHP also supports regular expressions using a Perl-compatible syntax using the PCRE functions. Those functions support non-greedy matching, assertions, conditional subpatterns, and a number of other features not supported by the POSIX-extended regular expression syntax.

Warning

These regular expression functions are not binary-safe. The PCRE functions are.

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 >= 4.0.0)

Regular expression match

int **ereg** (string pattern, string string [, array regs) \linebreak

Note: preg_match(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to **ereg()**.

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 the complete string matched.

Note: Up to (and including) PHP 4.1.0 *\$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(), eregi_replace(), and preg_match().

ereg_replace (PHP 3, PHP 4 >= 4.0.0)

Replace regular expression

string **ereg_replace** (string pattern, string replacement, string string) \linebreak

Note: preg_replace(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to **ereg_replace()**.

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 `\\digit`, 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.' */
?>
```

Example 3. Replace URLs with links

```
$text = ereg_replace("[[:alpha:]]+://[^<>[:space:]]+[[:alnum:]]/",
                    "<a href=\"\0\">\0</a>", $text);
```

See also `ereg()`, `eregi()`, `eregi_replace()`, `str_replace()`, and `preg_match()`.

eregi (PHP 3, PHP 4 >= 4.0.0)

case insensitive regular expression match

```
int eregi ( string pattern, string string [, array regs]) \linebreak
```

This function is identical to `ereg()` except that this ignores case distinction when matching alphabetic characters.

Example 1. eregi() example

```
if (eregi("z", $string)) {
    echo "'$string' contains a 'z' or 'Z'!";
}
```

See also `ereg()`, `ereg_replace()`, and `eregi_replace()`.

eregi_replace (PHP 3, PHP 4 >= 4.0.0)

replace regular expression case insensitive

```
string eregi_replace ( string pattern, string replacement, string string) \linebreak
```

This function is identical to `ereg_replace()` except that this ignores case distinction when matching alphabetic characters.

See also `ereg()`, `eregi()`, and `ereg_replace()`.

split (PHP 3, PHP 4 >= 4.0.0)

split string into array by regular expression

array **split** (string pattern, string string [, int limit]) \linebreak

Note: preg_split(), which uses a Perl-compatible regular expression syntax, is often a faster alternative to **split()**.

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

```
list($user,$pass,$uid,$gid,$extra)= split (":", $passwd_line, 5);
```

Note: If there are *n* occurrences of *pattern*, the returned array will contain *n+1* items. For example, if there is no occurrence of *pattern*, an array with only one element will be returned. Of course, this is also true if *string* is empty.

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: `preg_split()`, `spliti()`, `explode()`, `implode()`, `chunk_split()`, and `wordwrap()`.

spliti (PHP 4)

Split string into array by regular expression case insensitive

array **spliti** (string pattern, string string [, int limit]) \linebreak

This function is identical to `split()` except that this ignores case distinction when matching alphabetic characters.

See also **preg_spliti()**, `split()`, `explode()`, and `implode()`.

sql_regcase (PHP 3, PHP 4 >= 4.0.0)

Make regular expression for case insensitive match

string **sql_regcase** (string string) \linebreak

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.

LXXXIX. 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.

ftok (PHP 4 >= 4.2.0)

Convert a pathname and a project identifier to a System V IPC key

int **ftok** (string pathname, string proj) \linebreak

Warning

This function is currently not documented; only the argument list is available.

sem_acquire (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Acquire a semaphore

bool **sem_acquire** (int sem_identifier) \linebreak

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_get (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get a semaphore id

int **sem_get** (int key [, int max_acquire [, int perm]]) \linebreak

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_release (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Release a semaphore

bool **sem_release** (int sem_identifier) \linebreak

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.

sem_remove (PHP 4 >= 4.1.0)

Remove a semaphore

bool **sem_remove** (int sem_identifier) \linebreak

Returns: TRUE on success, FALSE on error.

sem_remove() removes the semaphore *sem_identifier* if it has been created by `sem_get()`, otherwise generates a warning.

After removing the semaphore, it is no more accessible.

See also: `sem_get()`, `sem_release()` and `sem_acquire()`.

Note: This function does not work on Windows systems. It was added on PHP 4.1.0.

shm_attach (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Creates or open a shared memory segment

int **shm_attach** (int key [, int memsize [, int perm]]) \linebreak

shm_attach() returns an id that 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. *Memsiz*e and *perm* will be ignored.

Note: This function does not work on Windows systems.

shm_detach (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Disconnects from shared memory segment

int **shm_detach** (int shm_identifier) \linebreak

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_get_var (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Returns a variable from shared memory

mixed **shm_get_var** (int id, int variable_key) \linebreak

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_put_var (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Inserts or updates a variable in shared memory

int **shm_put_var** (int shm_identifier, int variable_key, mixed variable) \linebreak

Inserts or updates a *variable* with a given *variable_key*. All variable-types are supported.

Note: This function does not work on Windows systems.

shm_remove (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Removes shared memory from Unix systems

```
int shm_remove ( int shm_identifier) \linebreak
```

Removes shared memory from Unix systems. All data will be destroyed.

Note: This function does not work on Windows systems.

shm_remove_var (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Removes a variable from shared memory

```
int shm_remove_var ( int id, int variable_key) \linebreak
```

Removes a variable with a given *variable_key* and frees the occupied memory.

Note: This function does not work on Windows systems.

XC. 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
<code>php3_sesam_oml</code>	Name of BS2000 PLAM library containing the loadable SESAM driver modules. Required for using SESAM functions. Example: <code>php3_sesam_oml \$.SYSLNK.SESAM-SQL.030</code>
<code>php3_sesam_configfile</code>	Name of SESAM application configuration file. Required for using SESAM functions. Example: <code>php3_sesam_configfile \$SESAM.SESAM.CONF.AW</code> It will usually contain a configuration like (see SESAM reference manual): <code>CNF=B</code> <code>NAM=K</code> <code>NOTYPE</code>

Directive	Meaning
php3_sesam_messagecatalog	<p>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:</p> <pre>php3_sesam_messagecatalog \$.SYSMES.SESAM-SQL.030</pre>

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 <i>offset</i> value, 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 adaptations 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	float
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_affected_rows (PHP 3 CVS only)

Get number of rows affected by an immediate query

```
int sesam_affected_rows ( string result_id) \linebreak
```

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_commit (PHP 3 CVS only)

Commit pending updates to the SESAM database

```
bool sesam_commit ( void) \linebreak
```

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_connect (PHP 3 CVS only)

Open SESAM database connection

```
bool sesam_connect ( string catalog, string schema, string user) \linebreak
```

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 "persistent" 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_diagnostic (PHP 3 CVS only)

Return status information for last SESAM call

array **sesam_diagnostic** (void) \linebreak

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($err["errmsg"])."</TH></TR>\n";
    if ($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_disconnect (PHP 3 CVS only)

Detach from SESAM connection

bool **sesam_disconnect** (void) \linebreak

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_errormsg (PHP 3 CVS only)

Returns error message of last SESAM call

string **sesam_errormsg** (void) \linebreak

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_execimm (PHP 3 CVS only)

Execute an "immediate" SQL-statement

string **sesam_execimm** (string query) \linebreak

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);
$error = sesam_diagnostic();
print ("sqlstate = ".$error["sqlstate"]."\n".
      "Affected rows = ".$error["rowcount"]." == ".
      sesam_affected_rows($result)."\n");
```

See also: **sesam_query()** and **sesam_affected_rows()**.

sesam_fetch_array (PHP 3 CVS only)

Fetch one row as an associative array

array **sesam_fetch_array** (string result_id [, int whence [, int offset]]) \linebreak

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_fetch_result (PHP 3 CVS only)

Return all or part of a query result

mixed **sesam_fetch_result** (string `result_id` [, int `max_rows`]) \linebreak

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 <code>\$arr["count"]</code>	number of columns in result set (or zero if this was an "immediate" query)
int <code>\$arr["rows"]</code>	number of rows in result set (between zero and <code>max_rows</code>)
bool <code>\$arr["truncated"]</code>	TRUE if the number of rows was at least <code>max_rows</code> , 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 <code>\$arr[col][row]</code>	result data for all the fields at row(<code>row</code>) and column(<code>col</code>), (where the integer index <code>row</code> is between 0 and <code>\$arr["rows"]-1</code> , and <code>col</code> is between 0 and <code>\$arr["count"]-1</code>). Fields may be empty, so you must check for the existence of a field by using the php <code>isset()</code> 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_fetch_row (PHP 3 CVS only)

Fetch one row as an array

```
array sesam_fetch_row ( string result_id [, int whence [, int offset]]) \linebreak
```

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)
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> (Zero-based. After fetch, the internal default is set to SESAM_SEEK_ABSOLUTE, and the internal offset value is auto-incremented)
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_field_array (PHP 3 CVS only)

Return meta information about individual columns in a result

array **sesam_field_array** (string *result_id*) \linebreak

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 <code>\$arr["count"]</code>	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 <code>\$arr[col]["name"]</code>	column name for column(<code>col</code>), where <code>col</code> is between 0 and <code>\$arr["count"]-1</code> . 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.
string <code>\$arr[col]["count"]</code>	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 <code>sesam_query()</code> description for a sample use of the "count" attribute.
string <code>\$arr[col]["type"]</code>	php variable type of the data for column(<code>col</code>), where <code>col</code> is between 0 and <code>\$arr["count"]-1</code> . The returned value can be one of <ul style="list-style-type: none"> • integer • float • 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.

Array Element	Contents
string \$arr[col]["sqltype"]	<p>SQL variable type of the column data for column(<i>col</i>), where <i>col</i> is between 0 and \$arr["count"]-1. The returned value can be one of</p> <ul style="list-style-type: none"> • "CHARACTER" • "VARCHAR" • "NUMERIC" • "DECIMAL" • "INTEGER" • "SMALLINT" • "FLOAT" • "REAL" • "DOUBLE" • "DATE" • "TIME" • "TIMESTAMP" <p>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.</p>
string \$arr[col]["length"]	<p>The SQL "length" attribute of the SQL variable in column(<i>col</i>), where <i>col</i> 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.</p>
string \$arr[col]["precision"]	<p>The "precision" attribute of the SQL variable in column(<i>col</i>), where <i>col</i> 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.</p>

Array Element	Contents
string \$arr[col]["scale"]	The "scale" attribute of the SQL variable in column(<i>col</i>), where <i>col</i> 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_field_name (PHP 3 CVS only)

Return one column name of the result set

```
int sesam_field_name ( string result_id, int index) \linebreak
```

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_free_result (PHP 3 CVS only)

Releases resources for the query

```
int sesam_free_result ( string result_id) \linebreak
```

Releases resources for the query associated with *result_id*. Returns `FALSE` on error.

sesam_num_fields (PHP 3 CVS only)

Return the number of fields/columns in a result set

```
int sesam_num_fields ( string result_id) \linebreak
```

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_query (PHP 3 CVS only)

Perform a SESAM SQL query and prepare the result

string **sesam_query** (string query [, bool scrollable]) \linebreak

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_rollback (PHP 3 CVS only)

Discard any pending updates to the SESAM database

bool **sesam_rollback** (void) \linebreak

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_seek_row (PHP 3 CVS only)

Set scrollable cursor mode for subsequent fetches

bool **sesam_seek_row** (string result_id, int whence [, int offset]) \linebreak

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)

Value	Constant	Meaning
4	SESAM_SEEK_ABSOLUTE	fetch absolute row number given as <i>offset</i> (Zero-based. After fetch, the default is set to SESAM_SEEK_ABSOLUTE, and the offset value is auto-incremented)
5	SESAM_SEEK_RELATIVE	fetch relative to current scroll position, where <i>offset</i> can be a positive or negative offset value (this also sets the default "offset" value for subsequent fetches).

offset is an optional parameter which is only evaluated (and required) if *whence* is either SESAM_SEEK_RELATIVE or SESAM_SEEK_ABSOLUTE.

sesam_settransaction (PHP 3 CVS only)

Set SESAM transaction parameters

```
bool sesam_settransaction ( int isolation_level, int read_only) \linebreak
```

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);
?>
```

XCI. 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.

Note: As of PHP 4.1.0, `$_SESSION` is available as global variable just like `$_POST`, `$_GET`, `$_REQUEST` and so on. Not like `$HTTP_SESSION_VARS`, `$_SESSION` is always global. Therefore, `global` should not be used for `$_SESSION`.

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_start();
if (isset($HTTP_SESSION_VARS['count'])) {
    $HTTP_SESSION_VARS['count']++;
}
else {
    $HTTP_SESSION_VARS['count'] = 0;
}
?>
```

Use of `$_SESSION` (or `$HTTP_SESSION_VARS` with PHP 4.0.6 or less) is recommended for security and code readability. With `$_SESSION` or `$HTTP_SESSION_VARS`, there is no need to use `session_register()/session_unregister()/session_is_registered()` functions. Users can access session variable like a normal variable.

Example 2. Registering a variable with `$_SESSION`.

```
<?php
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}
?>
```

Example 3. Unregistering a variable with `$_SESSION`.

```
<?php
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
unset($_SESSION['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. Since PHP must know which global variables are registered as session variables, users must register variables with `session_register()` function

while `$HTTP_SESSION_VARS/$_SESSION` does not need to use `session_register()`.

Caution

If you are using `$HTTP_SESSION_VARS/$_SESSION` and disable `register_globals`, do not use `session_register()`, `session_is_registered()` and `session_unregister()`.

If you enable `register_globals`, `session_unregister()` should be used since session variables are registered as global variables when session data is deserialized. Disabling `register_globals` is recommended for both security and performance reason.

Example 4. Registering a variable with `register_globals` enabled

```
<?php
if (!session_is_registered('count')) {
    session_register("count");
    $count = 0;
}
else {
    $count++;
}
?>
```

If both `track_vars` and `register_globals` are enabled, then the global variables and the `$HTTP_SESSION_VARS/$_SESSION` entries will reference the same value for already registered variables.

If user use `session_register()` to register session variable, `$HTTP_SESSION_VARS/$_SESSION` will not have these variable in array until it is loaded from session storage. (i.e. until next request)

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 5. Counting the number of hits of a single user

```
<?php
if (!session_is_registered('count')) {
    session_register('count');
    $count = 1;
}
else {
    $count++;
}
?>
```

```
Hello visitor, you have seen this page <?php echo $count; ?> times.<p>
```

```
<?php
# the <?php echo SID?> (<?=SID?> can be used if short tag is enabled)
# is necessary to preserve the session id
# in the case that the user has disabled cookies
?>
```

```
To continue, <A HREF="nextpage.php?<?php echo SID?>">click here</A>
```

The `<?=SID?>` is not necessary, if `--enable-trans-sid` was used to compile PHP.

Note: Non-relative URLs are assumed to point to external sites and hence don't append the `SID`, as it would be a security risk to leak the `SID` to a different server.

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`. If

`session.save_path`'s path depth is more than 2, garbage collection will not be performed.

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 (none/nocache/private/private_no_expire/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.
- `session.use_trans_sid` whether transparent sid support is enabled or not if enabled by compiling with `--enable-trans-sid`. Defaults to 1 (enabled).
- `url_rewriter.tags` specifies which html tags are rewritten to include session id if transparent sid

support is enabled. Defaults to `a=href`, `area=href`, `frame=src`, `input=src`, `form=fakeentry`

Note: Session handling was added in PHP 4.0.

session_cache_expire (PHP 4 >= 4.2.0)

Return current cache expire

```
int session_cache_expire ( [int new_cache_expire] ) \linebreak
```

session_cache_expire() returns current cache expire. If *new_cache_expire* is given, the current cache expire is replaced with *new_cache_expire*.

session_cache_limiter (PHP 4 >= 4.0.3)

Get and/or set the current cache limiter

```
string session_cache_limiter ( [string cache_limiter] ) \linebreak
```

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*.

In *private* mode, *Expire* header sent to the client, may cause confusion for some browser including Mozilla. You can avoid this problem with *private_no_expire* mode. *Expire* header is never sent to the client in this mode.

Note: *private_no_expire* was added in PHP 4.2.0dev.

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>";
?>
```

session_decode (PHP 4 >= 4.0.0)

Decodes session data from a string

```
bool session_decode ( string data) \linebreak
```

session_decode() decodes the session data in *data*, setting variables stored in the session.

session_destroy (PHP 4 >= 4.0.0)

Destroys all data registered to a session

```
bool session_destroy ( void) \linebreak
```

session_destroy() destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie.

This function returns `TRUE` on success and `FALSE` on failure to destroy the session data.

Example 1. Destroying a session

```
<?php

// Initialize the session.
// If you are using session_name("something"), don't forget it now!
session_start();
// Unset all of the session variables.
session_unset();
// Finally, destroy the session.
session_destroy();

?>
```

Example 2. Destroying a session with \$_SESSION

```
<?php

// Initialize the session.
// If you are using session_name("something"), don't forget it now!
session_start();
// Unset all of the session variables.
$_SESSION = array();
// Finally, destroy the session.
```

```
session_destroy();
```

```
?>
```

session_encode (PHP 4 >= 4.0.0)

Encodes the current session data as a string

```
string session_encode ( void) \linebreak
```

session_encode() returns a string with the contents of the current session encoded within.

session_get_cookie_params (PHP 4 >= 4.0.0)

Get the session cookie parameters

```
array session_get_cookie_params ( void) \linebreak
```

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.
- "secure" - The cookie should only be sent over secure connections. (This item was added in PHP 4.0.4.)

session_id (PHP 4 >= 4.0.0)

Get and/or set the current session id

```
string session_id ( [string id]) \linebreak
```

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_is_registered (PHP 4 >= 4.0.0)

Find out if a variable is registered in a session

bool **session_is_registered** (string *name*) \linebreak

session_is_registered() returns TRUE if there is a variable with the name *name* registered in the current session.

Note: If `$_SESSION` (or `$HTTP_SESSION_VARS` for PHP 4.0.6 or less) is used, use `isset()` to check a variable is registered in `$_SESSION`.

Caution

If you are using `$HTTP_SESSION_VARS/$_SESSION`, do not use `session_register()`, `session_is_registered()` and `session_unregister()`.

session_module_name (PHP 4 >= 4.0.0)

Get and/or set the current session module

string **session_module_name** ([string *module*]) \linebreak

session_module_name() returns the name of the current session module. If *module* is specified, that module will be used instead.

session_name (PHP 4 >= 4.0.0)

Get and/or set the current session name

string **session_name** ([string *name*]) \linebreak

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>";
?>
```

session_readonly (unknown)

Begin session - reinitializes freed variables, but no writeback on request end

```
void session_readonly ( void) \linebreak
```

Read in session data without locking the session data. Changing session data is not possible, but frameset performance will be improved.

session_register (PHP 4 >= 4.0.0)

Register one or more variables with the current session

```
bool session_register ( mixed name [, mixed ...]) \linebreak
```

session_register() accepts a variable number of arguments, any of which can be either a string holding the name of a variable or an array consisting of variable names or other arrays. For each name, **session_register()** registers the global variable with that name in the current session.

Caution

This registers a *global* variable. If you want to register a session variable inside a function, you need to make sure to make it global using **global()** or use the session arrays as noted below.

Caution

If you are using `$HTTP_SESSION_VARS/$_SESSION`, do not use **session_register()**, **session_is_registered()** and **session_unregister()**.

This function returns `TRUE` when all of the variables are successfully registered with the session.

If `session_start()` was not called before this function is called, an implicit call to `session_start()` with no parameters will be made.

You can also create a session variable by simply setting the appropriate member of the `$HTTP_SESSION_VARS` or `$_SESSION` (PHP >= 4.1.0) array.

```
$barney = "A big purple dinosaur.";
session_register("barney");

$HTTP_SESSION_VARS["zim"] = "An invader from another planet.";

# the auto-global $_SESSION array was introduced in PHP 4.1.0
$_SESSION["spongebob"] = "He's got square pants.";
```

Note: It is not currently possible to register resource variables in a session. For example, you can not create a connection to a database and store the connection id as a session variable and expect the connection to still be valid the next time the session is restored. PHP functions that return a resource are identified by having a return type of `resource` in their function definitions. A list of functions that return resources are available in the resource types appendix.

If `$_SESSION` (or `$HTTP_SESSION_VARS` for PHP 4.0.6 or less) is used, assign variable to `$_SESSION`. i.e. `$_SESSION['var'] = 'ABC'`;

See also `session_is_registered()` and `session_unregister()`.

session_save_path (PHP 4 >= 4.0.0)

Get and/or set the current session save path

string **session_save_path** ([string path]) \linebreak

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`.

session_set_cookie_params (PHP 4 >= 4.0.0)

Set the session cookie parameters

```
void session_set_cookie_params ( int lifetime [, string path [, string domain [, bool secure]]]) \linebreak
```

Set cookie parameters defined in the `php.ini` file. The effect of this function only lasts for the duration of the script.

Note: The `secure` parameter was added in PHP 4.0.4.

session_set_save_handler (PHP 4 >= 4.0.0)

Sets user-level session storage functions

```
void session_set_save_handler ( string open, string close, string read, string write, string destroy, string gc) \linebreak
```

`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.

Read function must return string value always to make save handler work as expected. Return empty string if there is no data to read. Return values from other handlers are converted to boolean expression. TRUE for success, FALSE for failure.

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(""); // Must return "" here.
    }
}

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_start (PHP 4 >= 4.0.0)

Initialize session data

```
bool session_start ( void) \linebreak
```

session_start() creates a session (or resumes the current one based on the session id being passed via a GET variable or a cookie).

If you want to use a named session, you must call `session_name()` before calling **session_start()**.

This function always returns `TRUE`.

Note: If you are using cookie-based sessions, you must call **session_start()** before anything is output to the browser.

session_start() will register internal output handler for URL rewriting when `trans-sid` is enabled. If a user uses `ob_gzhandler` or like with `ob_start()`, the order of output handler is important for proper output. For example, user must register `ob_gzhandler` before session start.

Note: Use of `zlib.output_compression` is recommended rather than `ob_gzhandler`

session_unregister (PHP 4 >= 4.0.0)

Unregister a variable from the current session

```
bool session_unregister ( string name) \linebreak
```

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: If `$_SESSION` (or `$HTTP_SESSION_VARS` for PHP 4.0.6 or less) is used, use `unset()` to unregister a session variable.

Caution

This function doesn't unset the corresponding global variable for *name*, it only prevents the variable from being saved as part of the session. You must call `unset()` to remove the corresponding global variable.

Caution

If you are using `$HTTP_SESSION_VARS/$_SESSION`, do not use `session_register()`, `session_is_registered()` and **`session_unregister()`**.

session_unset (PHP 4 >= 4.0.0)

Free all session variables

```
void session_unset ( void) \linebreak
```

The **`session_unset()`** function free's all session variables currently registered.

Note: If `$_SESSION` (or `$HTTP_SESSION_VARS` for PHP 4.0.6 or less) is used, use `unset()` to unregister session variable. i.e. `$_SESSION = array();`

session_write_close (PHP 4 >= 4.0.4)

Write session data and end session

```
void session_write_close ( void) \linebreak
```

End the current session and store session data.

Session data is usually stored after your script terminated without the need to call **`session_write_close()`**, but as session data is locked to prevent concurrent writes only one script may operate on a session at any time. When using framesets together with sessions you will experience the frames loading one by one due to this locking. You can reduce the time needed to load all the frames by ending the session as soon as all changes to session variables are done.

XCII. 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: This module is experimental. API and others are subject to be changed without notice.

In PHP 4.0.3, these functions were prefixed by `shm` rather than `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_close (PHP 4 >= 4.0.4)

Close shared memory block

```
int shmop_close ( int shmid) \linebreak
```

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.

shmop_delete (PHP 4 >= 4.0.4)

Delete shared memory block

```
int shmop_delete ( int shmid) \linebreak
```

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_open (PHP 4 >= 4.0.4)

Create or open shared memory block

```
int shmop_open ( int key, string flags, int mode, int size) \linebreak
```

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 SHM_RDONLY for shmat) use this flag when you need to open an existing shared memory segment for read only
- "c" for create (sets IPC_CREATE) use this flag when you need to create a new shared memory segment or if a segment with the same key exists, try to open it for read and write
- "w" for read & write access use this flag when you need to read and write to a shared memory segment, use this flag in most cases.
- "n" create a new memory segment (sets IPC_CREATE|IPC_EXCL) use this flag when you want to create a new shared memory segment but if one already exists with the same flag, fail. This is useful for security purposes, using this you can prevent race condition exploits.

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 (PHP 4 >= 4.0.4)

Read data from shared memory block

string **shmop_read** (int shmId, int start, int count) \linebreak

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_size (PHP 4 >= 4.0.4)

Get size of shared memory block

int **shmop_size** (int shmId) \linebreak

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 an 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_write (PHP 4 >= 4.0.4)

Write data into shared memory block

```
int shmop_write ( int shmid, string data, int offset) \linebreak
```

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.

XCIII. Shockwave Flash functions

PHP offers the ability to create Shockwave Flash files via Paul Haeberli's libswf module. You can download libswf at <ftp://ftp.sgi.com/cgi/graphics/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 ();  
?>
```

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.

For up to date SWF support take a look at the MING functions.

swf_actiongeturl (PHP 4 >= 4.0.0)

Get a URL from a Shockwave Flash movie

```
void swf_actiongeturl ( string url, string target) \linebreak
```

The **swf_actionGetUrl()** function gets the URL specified by the parameter *url* with the target *target*.

swf_actiongotoframe (PHP 4 >= 4.0.0)

Play a frame and then stop

```
void swf_actiongotoframe ( int framenummer) \linebreak
```

The **swf_actionGotoFrame()** function will go to the frame specified by *framenummer*, play it, and then stop.

swf_actiongotolabel (PHP 4 >= 4.0.0)

Display a frame with the specified label

```
void swf_actiongotolabel ( string label) \linebreak
```

The **swf_actionGotoLabel()** function displays the frame with the label given by the *label* parameter and then stops.

swf_actionnextframe (PHP 4 >= 4.0.0)

Go foward one frame

```
void swf_actionnextframe ( void) \linebreak
```

Go foward one frame.

swf_actionplay (PHP 4 >= 4.0.0)

Start playing the flash movie from the current frame

```
void swf_actionplay ( void) \linebreak
```

Start playing the flash movie from the current frame.

swf_actionprevframe (PHP 4 >= 4.0.0)

Go backwards one frame

```
void swf_actionprevframe ( void) \linebreak
```

swf_actionsettarget (PHP 4 >= 4.0.0)

Set the context for actions

```
void swf_actionsettarget ( string target) \linebreak
```

The `swf_actionSetTarget()` function sets the context for all actions. You can use this to control other flash movies that are currently playing.

swf_actionstop (PHP 4 >= 4.0.0)

Stop playing the flash movie at the current frame

```
void swf_actionstop ( void) \linebreak
```

Stop playing the flash movie at the current frame.

swf_actiontogglequality (PHP 4 >= 4.0.0)

Toggle between low and high quality

```
void swf_actiontogglequality ( void) \linebreak
```

Toggle the flash movie between high and low quality.

swf_actionwaitforframe (PHP 4 >= 4.0.0)

Skip actions if a frame has not been loaded

```
void swf_actionwaitforframe ( int framenummer, int skipcount) \linebreak
```

The `swf_actionWaitForFrame()` function will check to see if the frame, specified by the *framenummer* 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_addbuttonrecord (PHP 4 >= 4.0.0)

Controls location, appearance and active area of the current button

```
void swf_addbuttonrecord ( int states, int shapeid, int depth) \linebreak
```

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_addcolor (PHP 4 >= 4.0.0)

Set the global add color to the rgba value specified

```
void swf_addcolor ( float r, float g, float b, float a) \linebreak
```

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_closefile (PHP 4 >= 4.0.0)

Close the current Shockwave Flash file

```
void swf_closefile ( [int return_file]) \linebreak
```

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;
```

```

$dbh = @mysql_connect ($DBHOST, $DBUSER, $DBPASS);

if (!$dbh) {
    die (sprintf ("Error [%d]: %s",
                 mysql_errno (), mysql_error ()));
}

$stmt = "INSERT INTO swf_files (file) VALUES ('$data')";

$stmt = @mysql_query ($stmt, $dbh);

if (!$sth) {
    die (sprintf ("Error [%d]: %s",
                 mysql_errno (), mysql_error ()));
}

@mysql_free_result ($sth);
@mysql_close ($dbh);
}
?>

```

swf_definebitmap (PHP 4 >= 4.0.0)

Define a bitmap

```
void swf_definebitmap ( int objid, string image_name) \linebreak
```

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_definefont (PHP 4 >= 4.0.0)

Defines a font

```
void swf_definefont ( int fontid, string fontname) \linebreak
```

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_defineline (PHP 4 >= 4.0.0)

Define a line

```
void swf_defineline ( int objid, float x1, float y1, float x2, float y2, float width) \linebreak
```

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_definepoly (PHP 4 >= 4.0.0)

Define a polygon

```
void swf_definepoly ( int objid, array coords, int npoints, float width) \linebreak
```

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_definerect (PHP 4 >= 4.0.0)

Define a rectangle

```
void swf_definerect ( int objid, float x1, float y1, float x2, float y2, float width) \linebreak
```

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_definetext (PHP 4 >= 4.0.0)

Define a text string

```
void swf_definetext ( int objid, string str, int docenter) \linebreak
```

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_endbutton (PHP 4 >= 4.0.0)

End the definition of the current button

```
void swf_endbutton ( void) \linebreak
```

The **swf_endButton()** function ends the definition of the current button.

swf_enddoaction (PHP 4 >= 4.0.0)

End the current action

```
void swf_enddoaction ( void) \linebreak
```

Ends the current action started by the **swf_startdoaction()** function.

swf_endshape (PHP 4 >= 4.0.0)

Completes the definition of the current shape

```
void swf_endshape ( void) \linebreak
```

The **swf_endshape()** completes the definition of the current shape.

swf_endsymbol (PHP 4 >= 4.0.0)

End the definition of a symbol

```
void swf_endsymbol ( void) \linebreak
```

The **swf_endsymbol()** function ends the definition of a symbol that was started by the **swf_startsymbol()** function.

swf_fontsize (PHP 4 >= 4.0.0)

Change the font size

```
void swf_fontsize ( float size) \linebreak
```

The **swf_fontsize()** function changes the font size to the value given by the *size* parameter.

swf_fontslant (PHP 4 >= 4.0.0)

Set the font slant

```
void swf_fontslant ( float slant) \linebreak
```

Set the current font slant to the angle indicated by the *slant* parameter. Positive values create a forward slant, negative values create a negative slant.

swf_fontracking (PHP 4 >= 4.0.0)

Set the current font tracking

```
void swf_fontracking ( float tracking) \linebreak
```

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_getbitmapinfo (PHP 4 >= 4.0.0)

Get information about a bitmap

```
array swf_getbitmapinfo ( int bitmapid) \linebreak
```

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_getfontinfo (PHP 4 >= 4.0.0)

The height in pixels of a capital A and a lowercase x

```
array swf_getfontinfo ( void) \linebreak
```

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_getframe (PHP 4 >= 4.0.0)

Get the frame number of the current frame

```
int swf_getframe ( void) \linebreak
```

The **swf_getframe()** function gets the number of the current frame.

swf_labelframe (PHP 4 >= 4.0.0)

Label the current frame

```
void swf_labelframe ( string name) \linebreak
```

Label the current frame with the name given by the *name* parameter.

swf_lookat (PHP 4 >= 4.0.0)

Define a viewing transformation

```
void swf_lookat ( float view_x, float view_y, float view_z, float reference_x, float reference_y, float reference_z,  
float twist) \linebreak
```

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_modifyobject (PHP 4 >= 4.0.0)

Modify an object

```
void swf_modifyobject ( int depth, int how) \linebreak
```

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_mulcolor (PHP 4 >= 4.0.0)

Sets the global multiply color to the *rgba* value specified

```
void swf_mulcolor ( float r, float g, float b, float a) \linebreak
```

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_nextid (PHP 4 >= 4.0.0)

Returns the next free object id

```
int swf_nextid ( void) \linebreak
```

The `swf_nextid()` function returns the next available object id.

swf_oncondition (PHP 4 >= 4.0.0)

Describe a transition used to trigger an action list

```
void swf_oncondition ( int transition) \linebreak
```

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_openfile (PHP 4 >= 4.0.0)

Open a new Shockwave Flash file

```
void swf_openfile ( string filename, float width, float height, float framerate, float r, float g, float b) \linebreak
```

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_ortho (PHP 4)

Defines an orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho ( float xmin, float xmax, float ymin, float ymax, float zmin, float zmax) \linebreak
```

The **swf_ortho()** function defines a orthographic mapping of user coordinates onto the current viewport.

swf_ortho2 (PHP 4 >= 4.0.0)

Defines 2D orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho2 ( float xmin, float xmax, float ymin, float ymax) \linebreak
```

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 >= 4.0.0)

Define a perspective projection transformation

```
void swf_perspective ( float fovy, float aspect, float near, float far) \linebreak
```

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 to pretty.

swf_placeobject (PHP 4 >= 4.0.0)

Place an object onto the screen

```
void swf_placeobject ( int objid, int depth) \linebreak
```

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_polarview (PHP 4 >= 4.0.0)

Define the viewer's position with polar coordinates

```
void swf_polarview ( float dist, float azimuth, float incidence, float twist) \linebreak
```

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_popmatrix (PHP 4 >= 4.0.0)

Restore a previous transformation matrix

```
void swf_popmatrix ( void) \linebreak
```

The **swf_popmatrix()** function pushes the current transformation matrix back onto the stack.

swf_posround (PHP 4 >= 4.0.0)

Enables or Disables the rounding of the translation when objects are placed or moved

```
void swf_posround ( int round) \linebreak
```

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.

swf_pushmatrix (PHP 4 >= 4.0.0)

Push the current transformation matrix back unto the stack

```
void swf_pushmatrix ( void) \linebreak
```

The **swf_pushmatrix()** function pushes the current transformation matrix back onto the stack.

swf_removeobject (PHP 4 >= 4.0.0)

Remove an object

```
void swf_removeobject ( int depth) \linebreak
```

Removes the object at the depth specified by *depth*.

swf_rotate (PHP 4 >= 4.0.0)

Rotate the current transformation

```
void swf_rotate ( float angle, string axis) \linebreak
```

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_scale (PHP 4 >= 4.0.0)

Scale the current transformation

```
void swf_scale ( float x, float y, float z) \linebreak
```

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_setfont (PHP 4 >= 4.0.0)

Change the current font

```
void swf_setfont ( int fontid) \linebreak
```

The **swf_setfont()** sets the current font to the value given by the *fontid* parameter.

swf_setframe (PHP 4 >= 4.0.0)

Switch to a specified frame

```
void swf_setframe ( int framenum) \linebreak
```

The `swf_setframe()` changes the active frame to the frame specified by *framenumber*.

swf_shapearc (PHP 4 >= 4.0.0)

Draw a circular arc

```
void swf_shapearc ( float x, float y, float r, float ang1, float ang2) \linebreak
```

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_shapecurveto (PHP 4 >= 4.0.0)

Draw a quadratic bezier curve between two points

```
void swf_shapecurveto ( float x1, float y1, float x2, float y2) \linebreak
```

The `swf_shapecurveto()` function draws a quadratic bezier curve from the current location, though 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 >= 4.0.0)

Draw a cubic bezier curve

```
void swf_shapecurveto3 ( float x1, float y1, float x2, float y2, float x3, float y3) \linebreak
```

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_shapefillbitmapclip (PHP 4 >= 4.0.0)

Set current fill mode to clipped bitmap

```
void swf_shapefillbitmapclip ( int bitmapid) \linebreak
```

Sets the fill to bitmap clipped, empty spaces will be filled by the bitmap given by the *bitmapid* parameter.

swf_shapefillbitmaptile (PHP 4 >= 4.0.0)

Set current fill mode to tiled bitmap

```
void swf_shapefillbitmaptile ( int bitmapid) \linebreak
```

Sets the fill to bitmap tile, empty spaces will be filled by the bitmap given by the *bitmapid* parameter (tiled).

swf_shapefilloff (PHP 4 >= 4.0.0)

Turns off filling

```
void swf_shapefilloff ( void) \linebreak
```

The `swf_shapeFillOff()` function turns off filling for the current shape.

swf_shapefillsolid (PHP 4 >= 4.0.0)

Set the current fill style to the specified color

```
void swf_shapefillsolid ( float r, float g, float b, float a) \linebreak
```

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_shapelinesolid (PHP 4 >= 4.0.0)

Set the current line style

```
void swf_shapelinesolid ( float r, float g, float b, float a, float width) \linebreak
```

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_shapelineto (PHP 4 >= 4.0.0)

Draw a line

```
void swf_shapelineto ( float x, float y) \linebreak
```

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_shapemoveto (PHP 4 >= 4.0.0)

Move the current position

```
void swf_shapemoveto ( float x, float y) \linebreak
```

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_showframe (PHP 4 >= 4.0.0)

Display the current frame

```
void swf_showframe ( void) \linebreak
```

The **swf_showframe** function will output the current frame.

swf_startbutton (PHP 4 >= 4.0.0)

Start the definition of a button

```
void swf_startbutton ( int objid, int type) \linebreak
```

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_startdoaction (PHP 4 >= 4.0.0)

Start a description of an action list for the current frame

```
void swf_startdoaction ( void) \linebreak
```

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_startshape (PHP 4 >= 4.0.0)

Start a complex shape

```
void swf_startshape ( int objid) \linebreak
```

The **swf_startshape()** function starts a complex shape, with an object id given by the *objid* parameter.

swf_startsymbol (PHP 4 >= 4.0.0)

Define a symbol

```
void swf_startsymbol ( int objid) \linebreak
```

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_textwidth (PHP 4 >= 4.0.0)

Get the width of a string

```
float swf_textwidth ( string str) \linebreak
```

The **swf_textwidth()** function gives the width of the string, *str*, in pixels, using the current font and font size.

swf_translate (PHP 4 >= 4.0.0)

Translate the current transformations

```
void swf_translate ( float x, float y, float z) \linebreak
```

The **swf_translate()** function translates the current transformation by the *x*, *y*, and *z* values given.

swf_viewport (PHP 4 >= 4.0.0)

Select an area for future drawing

```
void swf_viewport ( float xmin, float xmax, float ymin, float ymax) \linebreak
```

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.

XCIV. SNMP functions

In order to use the SNMP functions on Unix you need to install the UCD SNMP (<http://net-snmp.sourceforge.net/>) 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.

snmp_get_quick_print (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Fetch the current value of the UCD library's quick_print setting

```
bool snmp_get_quick_print ( void) \linebreak
```

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 off, 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 >= 4.0.0)

Set the value of quick_print within the UCD SNMP library

```
void snmp_set_quick_print ( bool quick_print) \linebreak
```

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.

snmpget (PHP 3, PHP 4 >= 4.0.0)

Fetch an SNMP object

string **snmpget** (string hostname, string community, string object_id [, int timeout [, int retries]]) \linebreak

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");
```

snmprealwalk (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Return all objects including their respective object id withing the specified one

array **snmprealwalk** (string host, string community, string object_id [, int timeout [, int retries]]) \linebreak

Warning

This function is currently not documented; only the argument list is available.

snmpset (PHP 3>= 3.0.12, PHP 4 >= 4.0.0)

Set an SNMP object

bool **snmpset** (string hostname, string community, string object_id, string type, mixed value [, int timeout [, int retries]]) \linebreak

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 >= 4.0.0)

Fetch all the SNMP objects from an agent

array **snmpwalk** (string hostname, string community, string object_id [, int timeout [, int retries]]) \linebreak

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 >= 4.0.0)

Query for a tree of information about a network entity

array **snmpwalkoid** (string hostname, string community, string object_id [, int timeout [, int retries]]) \linebreak

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";  
}
```

XCV. Socket functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

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 `pfssockopen()`.

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'.

```
#!/usr/local/bin/php -q
<?php
error_reporting (E_ALL);

/* Allow the script to hang around waiting for connections. */
set_time_limit (0);

/* Turn on implicit output flushing so we see what we're getting
 * as it comes in. */
ob_implicit_flush ();

$address = '192.168.1.53';
$port = 10000;

if (($sock = socket_create (AF_INET, SOCK_STREAM, 0)) < 0) {
    echo "socket_create() failed: reason: " . socket_strerror ($sock) . "\n";
}

if (($ret = socket_bind ($sock, $address, $port)) < 0) {
```

```

    echo "socket_bind() failed: reason: " . socket_strerror ($ret) . "\n";
}

if (($ret = socket_listen ($sock, 5)) < 0) {
    echo "socket_listen() failed: reason: " . socket_strerror ($ret) . "\n";
}

do {
    if (($msgsock = socket_accept($sock)) < 0) {
        echo "socket_accept() failed: reason: " . socket_strerror ($msgsock) . "\n";
        break;
    }
    /* Send instructions. */
    $msg = "\nWelcome to the PHP Test Server. \n" .
        "To quit, type 'quit'. To shut down the server type 'shutdown'.\n";
    socket_write($msgsock, $msg, strlen($msg));

    do {
        if (FALSE === ($buf = socket_read ($msgsock, 2048))) {
            echo "socket_read() failed: reason: " . socket_strerror ($ret) . "\n";
            break 2;
        }
        if (!$buf = trim ($buf)) {
            continue;
        }
        if ($buf == 'quit') {
            break;
        }
        if ($buf == 'shutdown') {
            socket_close ($msgsock);
            break 2;
        }
        $talkback = "PHP: You said '$buf'.\n";
        socket_write ($msgsock, $talkback, strlen ($talkback));
        echo "$buf\n";
    } while (true);
    socket_close ($msgsock);
} while (true);

socket_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.example.com');

/* Create a TCP/IP socket. */
$socket = socket_create (AF_INET, SOCK_STREAM, 0);
if ($socket < 0) {
    echo "socket_create() failed: reason: " . socket_strerror ($socket) . "\n";
} else {
    echo "OK.\n";
}

echo "Attempting to connect to '$address' on port '$service_port'...";
$result = socket_connect ($socket, $address, $service_port);
if ($result < 0) {
    echo "socket_connect() failed.\nReason: ($result) " . socket_strerror($result) . "\n";
} else {
    echo "OK.\n";
}

$in = "HEAD / HTTP/1.0\r\n\r\n";
$out = "";

echo "Sending HTTP HEAD request...";
socket_write ($socket, $in, strlen ($in));
echo "OK.\n";

echo "Reading response:\n\n";
while ($out = socket_read ($socket, 2048)) {
    echo $out;
}

echo "Closing socket...";
socket_close ($socket);
echo "OK.\n\n";
?>

```

socket_accept (PHP 4 >= 4.1.0)

Accepts a connection on a socket

```
int socket_accept ( resource socket) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

After the socket *socket* has been created using `socket_create()`, bound to a name with `socket_bind()`, and told to listen for connections with `socket_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, `socket_accept()` will block until a connection becomes present. If *socket* has been made non-blocking using `socket_set_blocking()` or `socket_set_nonblock()`, an error code will be returned.

The socket descriptor returned by `socket_accept()` 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 `socket_strerror()` to get a textual explanation of the error.

See also `socket_bind()`, `socket_connect()`, `socket_listen()`, `socket_create()`, and `socket_strerror()`.

socket_bind (PHP 4 >= 4.1.0)

Binds a name to a socket

```
int socket_bind ( resource socket, string address [, int port]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

`socket_bind()` binds the name given in *address* to the socket described by *socket*, which must be a valid socket descriptor created with `socket_create()`.

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 `socket_strerror()` to get a textual explanation of the error.

See also `socket_connect()`, `socket_listen()`, `socket_create()`, and `socket_strerror()`.

socket_close (PHP 4 >= 4.1.0)

Closes a socket descriptor

bool **socket_close** (resource *socket*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

socket_close() closes the file (or socket) descriptor given by *socket*.

Note that **socket_close()** should not be used on PHP file descriptors created with `fopen()`, `popen()`, `fsockopen()`, or **psockopen()**; it is meant for sockets created with `socket_create()` or `socket_accept()`.

Returns `TRUE` on success, or `FALSE` if an error occurs (i.e., *socket* is invalid).

See also `socket_bind()`, `socket_listen()`, `socket_create()`, and `socket_strerror()`.

socket_connect (PHP 4 >= 4.1.0)

Initiates a connection on a socket

bool **socket_connect** (resource *socket*, string *address* [, int *port*]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Initiates a connection using the socket descriptor *socket*, which must be a valid socket descriptor created with `socket_create()`.

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 `TRUE` on success, `FALSE` on failure. This code may be passed to `socket_strerror()` to get a textual explanation of the error.

See also `socket_bind()`, `socket_listen()`, `socket_create()`, and `socket_strerror()`.

socket_create (PHP 4 >= 4.1.0)

Create a socket (endpoint for communication)

resource **socket_create** (int domain, int type, int protocol) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

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 `socket_strerror()` to get a textual explanation of the error.

For more information on the usage of `socket_create()`, as well as on the meanings of the various parameters, see the Unix man page `socket(2)`.

See also `socket_accept()`, `socket_bind()`, `socket_connect()`, `socket_listen()`, and `socket_strerror()`.

socket_create_listen (PHP 4 >= 4.1.0)

Opens a socket on port to accept connections

resource **socket_create_listen** (int port [, int backlog]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_create_pair (PHP 4 >= 4.1.0)

Creates a pair of indistinguishable sockets and stores them in fds.

bool **socket_create_pair** (int domain, int type, int protocol, array &fd) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_alloc (4.1.0 - 4.1.2 only)

Allocates a new file descriptor set

resource **socket_fd_alloc** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_clear (4.1.0 - 4.1.2 only)

Clears (a) file descriptor(s) from a set

bool **socket_fd_clear** (resource set, mixed socket) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_free (4.1.0 - 4.1.2 only)

Deallocates a file descriptor set

bool **socket_fd_free** (resource set) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_isset (4.1.0 - 4.1.2 only)

Checks to see if a file descriptor is set within the file descriptor set

```
bool socket_fd_isset ( resource set, resource socket) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_set (4.1.0 - 4.1.2 only)

Adds (a) file descriptor(s) to a set

```
bool socket_fd_set ( resource set, mixed socket) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_fd_zero (4.1.0 - 4.1.2 only)

Clears a file descriptor set

```
bool socket_fd_zero ( resource set) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_getopt (PHP 4 >= 4.1.0)

Gets socket options for the socket

mixed **socket_getopt** (resource socket, int level, int optname) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_getpeername (PHP 4 >= 4.1.0)

Given an fd, stores a string representing sa.sin_addr and the value of sa.sin_port into addr and port describing the remote side of a socket

bool **socket_getpeername** (resource socket, string &addr [, int &port]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_getsockname (PHP 4 >= 4.1.0)

Given an fd, stores a string representing sa.sin_addr and the value of sa.sin_port into addr and port describing the local side of a socket

```
bool socket_getsockname ( resource socket, string &addr [, int &port]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_add (PHP 4 >= 4.1.0)

Adds a new vector to the scatter/gather array

```
bool socket_iovec_add ( resource iovvec, int iov_len) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_alloc (PHP 4 >= 4.1.0)

...]) Builds a 'struct iovec' for use with sendmsg, recvmsg, writev, and readv

resource **socket_iovec_alloc** (int num_vectors [, int]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_delete (PHP 4 >= 4.1.0)

Deletes a vector from an array of vectors

bool **socket_iovec_delete** (resource iovec, int iov_pos) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_fetch (PHP 4 >= 4.1.0)

Returns the data held in the iovec specified by iovec_id[iovec_position]

string **socket_iovec_fetch** (resource iovec, int iovec_position) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_free (PHP 4 >= 4.1.0)

Frees the iovec specified by iovec_id

bool **socket_iovec_free** (resource iovec) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_iovec_set (PHP 4 >= 4.1.0)

Sets the data held in `iovec_id[iovec_position]` to `new_val`

```
bool socket_iovec_set ( resource iovec, int iovec_position, string new_val) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_last_error (PHP 4 >= 4.1.0)

Returns/Clears the last error on the socket

```
int socket_last_error ( resource socket) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_listen (PHP 4 >= 4.1.0)

Listens for a connection on a socket

```
int socket_listen ( resource socket, int backlog) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

After the socket *socket* has been created using `socket_create()` and bound to a name with `socket_bind()`, it may be told to listen for incoming connections on *socket*. A maximum of *backlog* incoming connections will be queued for processing.

`socket_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 `socket_strerror()` to get a textual explanation of the error.

See also `socket_accept()`, `socket_bind()`, `socket_connect()`, `socket_create()`, and `socket_strerror()`.

socket_read (PHP 4 >= 4.1.0)

Reads from a socket

```
string socket_read ( resource socket_des, int length [, int type]) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The function `socket_read()` reads from socket *socket_des* created by the `socket_accept()` function the number of bytes set by *length*. Otherwise you can use `\n`, `\t` or `\0` to end reading. Returns data, or `FALSE` if `socket_read()` failed.

Optional *type* parameter is a named constant:

- `PHP_BINARY_READ` - use the system `socket_read()` (Default in PHP >= 4.1.0)
- `PHP_NORMAL_READ` - reading stops at `\n` or `\r`. (Default in PHP <= 4.0.6)

See also `socket_accept()`, `socket_bind()`, `socket_connect()`, `socket_listen()`, `socket_strerror()`, and `socket_write()`.

socket_readv (PHP 4 >= 4.1.0)

Reads from an fd, using the scatter-gather array defined by `iovec_id`

bool **socket_readv** (resource socket, resource `iovec_id`) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_recv (PHP 4 >= 4.1.0)

Receives data from a connected socket

string **socket_recv** (resource socket, int len, int flags) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_recvfrom (PHP 4 >= 4.1.0)

Receives data from a socket, connected or not

int **socket_recvfrom** (resource socket, string &buf, int len, int flags, string &name [, int &port]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_recvmsg (PHP 4 >= 4.1.0)

Used to receive messages on a socket, whether connection-oriented or not

bool **socket_recvmsg** (resource socket, resource iovec, array &control, int &controllen, int &flags, string &addr [, int &port]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_select (PHP 4 >= 4.1.0)

Runs the select() system call on the sets mentioned with a timeout specified by tv_sec and tv_usec

int **socket_select** (resource read_fd, resource write_fd, resource except_fd, int tv_sec [, int tv_usec]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_send (PHP 4 >= 4.1.0)

Sends data to a connected socket

int **socket_send** (resource socket, string buf, int len, int flags) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_sendmsg (PHP 4 >= 4.1.0)

Sends a message to a socket, regardless of whether it is connection-oriented or not

bool **socket_sendmsg** (resource socket, resource iovec, int flags, string addr [, int port]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_sendto (PHP 4 >= 4.1.0)

Sends a message to a socket, whether it is connected or not

int **socket_sendto** (resource socket, string buf, int len, int flags, string addr [, int port]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_set_nonblock (PHP 4 >= 4.1.0)

Sets nonblocking mode for file descriptor fd

bool **socket_set_nonblock** (resource socket) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_setopt (PHP 4 >= 4.1.0)

Sets socket options for the socket

bool **socket_setopt** (resource socket, int level, int optname, int) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_shutdown (PHP 4 >= 4.1.0)

Shuts down a socket for receiving, sending, or both.

bool **socket_shutdown** (resource socket [, int how]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

socket_strerror (PHP 4 >= 4.1.0)

Return a string describing a socket error

string **socket_strerror** (int *errno*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

socket_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 **socket_strerror()**, and it tells you what happened.

Example 1. socket_strerror() example

```

<?php
if (($socket = socket_create (AF_INET, SOCK_STREAM, 0)) < 0) {
    echo "socket_create() failed: reason: " . socket_strerror ($socket) . "\n";
}

if (($ret = socket_bind ($socket, '127.0.0.1', 80)) < 0) {
    echo "socket_bind() failed: reason: " . socket_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 [socket_accept\(\)](#), [socket_bind\(\)](#), [socket_connect\(\)](#), [socket_listen\(\)](#), and [socket_create\(\)](#).

socket_write (PHP 4 >= 4.1.0)

Write to a socket

int **socket_write** (resource *socket_des*, string *&buffer*, int *length*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

The function **socket_write()** writes to the socket *socket_des* from *&buffer* the number of bytes set by *length*.

See also `socket_accept()`, `socket_bind()`, `socket_connect()`, `socket_listen()`, `socket_read()`, and `socket_strerror()`.

socket_writev (PHP 4 >= 4.1.0)

Writes to a file descriptor, *fd*, using the scatter-gather array defined by *iovec_id*

bool **socket_writev** (resource *socket*, resource *iovec_id*) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

XCVI. 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.

For even more powerful string handling and manipulating functions take a look at the POSIX regular expression functions and the Perl compatible regular expression functions

addslashes (PHP 4 >= 4.0.0)

Quote string with slashes in a C style

string **addslashes** (string str, string charlist) \linebreak

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 careful if you choose to escape characters `0`, `a`, `b`, `f`, `n`, `r`, `t` and `v`. They will be converted to `\0`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t` and `\v`. In PHP `\0` (`NULL`), `\r` (carriage return), `\n` (newline) and `\t` (tab) are predefined escape sequences, while in C all of these are predefined escape sequences.

charlist like `"\0..\37"`, which would escape all characters with ASCII code between 0 and 31.

Example 1. addslashes() example

```
$escaped = addslashes($not_escaped, "\0..\37!@\177..\377");
```

When you define a sequence of characters in the *charlist* argument make sure that you know what characters come between the characters that you set as the start and end of the range.

```
echo addslashes('foo[ ]', 'A..z');
// output:  \f\o\o\[ \]
// All upper and lower-case letters will be escaped
// ... but so will the [\]^_` and any tabs, line
// feeds, carriage returns, etc.
```

Also, if the first character in a range has a lower ASCII value than the second character in the range, no range will be constructed. Only the start, end and period characters will be escaped. Use the `ord()` function to find the ASCII value for a character.

```
echo addslashes("zoo['.']", 'z..A');
// output:  \zoo['\.']
```

See also `stripslashes()`, `stripslashes()`, `htmlspecialchars()`, and `quotemeta()`.

addslashes (PHP 3, PHP 4 >= 4.0.0)

Quote string with slashes

string **addslashes** (string *str*) \linebreak

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).

Note: `magic_quotes_gpc` is ON by default.

See also `stripslashes()`, `htmlspecialchars()`, and `quotemeta()`.

bin2hex (PHP 3 >= 3.0.9, PHP 4 >= 4.0.0)

Convert binary data into hexadecimal representation

string **bin2hex** (string *str*) \linebreak

Returns an ASCII string containing the hexadecimal representation of *str*. The conversion is done byte-wise with the high-nibble first.

See also `pack()` and `unpack()`.

chop (PHP 3, PHP 4 >= 4.0.0)

Alias of `rtrim()`

This function is an alias of `rtrim()`.

Note: `chop()` is different than the Perl `chop()` function, which removes the last character in the string.

chr (PHP 3, PHP 4 >= 4.0.0)

Return a specific character

string **chr** (int *ascii*) \linebreak

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);
```

You can find an ASCII-table over here: <http://www.mindspring.com/~jcl/serial/Resources/ASCII.html>.

This function complements ord(). See also sprintf() with a format string of %c.

chunk_split (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Split a string into smaller chunks

string **chunk_split** (string body [, int chunklen [, string end]]) \linebreak

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 *end* (defaults to "\r\n") every *chunklen* characters (defaults to 76). 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));
```

See also explode(), split() and wordwrap().

convert_cyr_string (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Convert from one Cyrillic character set to another

string **convert_cyr_string** (string str, string from, string to) \linebreak

This function returns the given string converted 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 >= 4.0.0)

Return information about characters used in a string

mixed **count_chars** (string string [, int mode]) \linebreak

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.

crc32 (PHP 4)

Calculates the crc32 polynomial of a string

int **crc32** (string str) \linebreak

Generates the cyclic redundancy checksum polynomial of 32-bit lengths of the *str*. This is usually used to validate the integrity of data being transmitted.

See also: md5()

crypt (PHP 3, PHP 4 >= 4.0.0)

One-way string encryption (hashing)

string **crypt** (string *str* [, string *salt*]) \linebreak

crypt() will return an encrypted string using the standard Unix DES-based encryption algorithm or alternative algorithms that may be available on the system. Arguments are a string to be encrypted and an optional 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-based 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 two character 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 two character salt applies to your system or the longer twelve character 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 security.

The standard DES-based encryption **crypt()** returns the salt as the first two characters of the output. It also only uses the first eight characters of *str*, so longer strings that start with the same eight characters will generate the same result (when the same salt is used).

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-based encryption with a two character salt
- `CRYPT_EXT_DES` - Extended DES-based encryption with a nine character salt
- `CRYPT_MD5` - MD5 encryption with a twelve character salt starting with \$1\$
- `CRYPT_BLOWFISH` - Blowfish encryption with a sixteen character salt starting with \$2\$

Note: There is no decrypt function, since **crypt()** uses a one-way algorithm.

Example 1. crypt() examples

```
<?php
$password = crypt("MylsTpassword"); # let salt be generated

# You should pass the entire results of crypt() as the salt for comparing a
# password, to avoid problems when different hashing algorithms are used. (As
# it says above, standard DES-based password hashing uses a 2-character salt,
# but MD5-based hashing uses 12.)
if (crypt($user_input,$password) == $password) {
    echo "Password verified!";
}
?>
```

See also `md5()` and the Mcrypt extension.

echo (unknown)

Output one or more strings

echo (string arg1 [, string argn...]) \linebreak

Outputs all parameters.

echo() is not actually a function (it is a language construct) so you are not required to use parentheses with it. In fact, if you want to pass more than one parameter to `echo`, you must not enclose the parameters within parentheses. It is not possible to use **echo()** in a variable function context, but you can use `print()` instead.

Example 1. echo() examples

```
<?php
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.";

echo "escaping characters is done \"Like this\"."

//You can use variables inside of an echo statement
$foo = "foobar";
$bar = "barbaz";

echo "foo is $foo"; // foo is foobar

// Using single quotes will print the variable name, not the value
echo 'foo is $foo'; // foo is $foo

// If you are not using any other characters, you can just echo variables
echo $foo;           // foobar
echo $foo,$bar;     // foobarbarbaz

echo <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
```

```
// because echo is not a function, following code is invalid.
($some_var) ? echo('true'): echo('false');

// However, the following examples will work:
($some_var) ? print('true'): print('false'); // print is a function
echo $some_var ? 'true': 'false'; // changing the statement around
?>
```

echo() also has a shortcut syntax, where you can immediately follow the opening tag with an equals sign.

```
I have <?=$foo?> foo.
```

Note: This short syntax only works with the `short_open_tag` configuration setting enabled.

See also: `print()`, `printf()`, and `flush()`.

explode (PHP 3, PHP 4 >= 4.0.0)

Split a string by string

array **explode** (string separator, string string [, int limit]) \linebreak

Returns an array of strings, each of which is a substring of *string* formed by splitting it on boundaries formed by the string *separator*. If *limit* is set, the returned array will contain a maximum of *limit* elements with the last element containing the rest of *string*.

If *separator* is an empty string (""), **explode()** will return `FALSE`. If *separator* contains a value that is not contained in *string*, then **explode()** will return an array containing *string*.

Note: The *limit* parameter was added in PHP 4.0.1

Example 1. explode() examples

```
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);

$data = "foo:*:1023:1000:~/home/foo:/bin/sh";
list($user,$pass,$uid,$gid,$gecos,$home,$shell) = explode(":",$data);
```

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 `preg_split()`, `spliti()`, `split()`, and `implode()`.

get_html_translation_table (PHP 4 >= 4.0.0)

Returns the translation table used by `htmlspecialchars()` and `htmlentities()`

```
string get_html_translation_table ( int table [, int quote_style] ) \linebreak
```

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);
$original = strtr($encoded, $trans);
```

The content of `$original` would be: "Hallo & <Frau> & Krämer".

See also: `htmlspecialchars()`, `htmlentities()`, `strtr()`, and `array_flip()`.

get_meta_tags (PHP 3 >= 3.0.4, PHP 4 >= 4.0.0)

Extracts all meta tag content attributes from a file and returns an array

array **get_meta_tags** (string filename [, int use_include_path]) \linebreak

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.

hebrew (PHP 3, PHP 4 >= 4.0.0)

Convert logical Hebrew text to visual text

string **hebrew** (string hebrew_text [, int max_chars_per_line]) \linebreak

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 >= 4.0.0)

Convert logical Hebrew text to visual text with newline conversion

string **hebrevc** (string hebrew_text [, int max_chars_per_line]) \linebreak

This function is similar to `hebrevc()` 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 `hebrevc()`

htmlspecialchars (PHP 3, PHP 4 >= 4.0.0)

Convert all applicable characters to HTML entities

string **htmlspecialchars** (string string [, int quote_style [, string charset]]) \linebreak

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 as default. Support for the optional second argument was added in PHP 3.0.17 and PHP 4.0.3.

Like `htmlspecialchars()`, it takes an optional third argument which defines character set used in conversion. Support for this argument was added in PHP 4.1.0.

There is no reverse of this function. However, you can create one on your own. Here is an example of how to do this.

Example 1. Reverse of htmlspecialchars()

```
<?php
function unhtmlspecialchars ($string)
{
    $trans_tbl = get_html_translation_table (HTML_ENTITIES);
    $trans_tbl = array_flip ($trans_tbl);
    return strtr ($string, $trans_tbl);
}
?>
```

See also `htmlspecialchars()` and `nl2br()`.

htmlspecialchars (PHP 3, PHP 4 >= 4.0.0)

Convert special characters to HTML entities

string **htmlspecialchars** (string string [, int quote_style [, string charset]]) \linebreak

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()`. Support for the optional second argument was added in PHP 3.0.17 and PHP 4.0.3.

The third argument defines character set used in conversion. The default character set is ISO-8859-1. Support for this third argument was added in PHP 4.1.0.

See also `htmlentities()` and `nl2br()`.

implode (PHP 3, PHP 4 >= 4.0.0)

Join array elements with a string

string **implode** (string glue, array pieces) \linebreak

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 >= 4.0.0)

Join array elements with a string

```
string join ( string glue, array pieces) \linebreak
```

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)

Calculate Levenshtein distance between two strings

```
int levenshtein ( string str1, string str2) \linebreak
int levenshtein ( string str1, string str2, int cost_ins, int cost_rep, int cost_del) \linebreak
int levenshtein ( string str1, string str2, function cost) \linebreak
```

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 (PHP 4 >= 4.0.5)

Get numeric formatting information

array **localeconv** (void) \linebreak

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
<code>decimal_point</code>	Decimal point character
<code>thousands_sep</code>	Thousands separator
<code>grouping</code>	Array containing numeric groupings
<code>int_curr_symbol</code>	International currency symbol (i.e. USD)
<code>currency_symbol</code>	Local currency symbol (i.e. \$)
<code>mon_decimal_point</code>	Monetary decimal point character
<code>mon_thousands_sep</code>	Monetary thousands separator
<code>mon_grouping</code>	Array containing monetary groupings
<code>positive_sign</code>	Sign for positive values

Array element	Description
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	<p>0 Parentheses surround the quantity and currency_symbol</p> <p>1 The sign string precedes the quantity and currency_symbol</p> <p>2 The sign string succeeds the quantity and currency_symbol</p> <p>3 The sign string immediately precedes the currency_symbol</p> <p>4 The sign string immediately succeeds the currency_symbol</p>
n_sign_posn	<p>0 Parentheses surround the quantity and currency_symbol</p> <p>1 The sign string precedes the quantity and currency_symbol</p> <p>2 The sign string succeeds the quantity and currency_symbol</p> <p>3 The sign string immediately precedes the currency_symbol</p> <p>4 The sign string immediately succeeds the currency_symbol</p>

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.

See also: setlocale().

ltrim (PHP 3, PHP 4 >= 4.0.0)

Strip whitespace from the beginning of a string

string **ltrim** (string *str* [, string *charlist*]) \linebreak

Note: The second parameter was added in PHP 4.1.0

This function returns a string with whitespace stripped from the beginning of *str*. Without the second parameter, **ltrim()** will strip these characters:

- " " (ASCII 32 (0x20)), an ordinary space.
- "\t" (ASCII 9 (0x09)), a tab.
- "\n" (ASCII 10 (0x0A)), a new line (line feed).
- "\r" (ASCII 13 (0x0D)), a carriage return.
- "\0" (ASCII 0 (0x00)), the NUL-byte.
- "\x0B" (ASCII 11 (0x0B)), a vertical tab.

You can also specify the characters you want to strip, by means of the *charlist* parameter. Simply list all characters that you want to be stripped. With `.` you can specify a range of characters.

Example 1. Usage example of `ltrim()`

```
<?php

$text = "\t\tThese are a few words :) ... ";
$trimmed = ltrim($text);
// $trimmed = "These are a few words :) ... "
$trimmed = ltrim($text, " \t.");
// $trimmed = "These are a few words :) ... "
$clean = ltrim($binary, "\0x00..\0x1F");
// trim the ASCII control characters at the beginning of $binary
// (from 0 to 31 inclusive)

?>
```

See also `trim()` and `rtrim()`.

md5 (PHP 3, PHP 4 >= 4.0.0)

Calculate the md5 hash of a string

string **md5** (string *str*) \linebreak

Calculates the MD5 hash of *str* using the RSA Data Security, Inc. MD5 Message-Digest Algorithm (<http://www.faqs.org/rfcs/rfc1321.html>), and returns that hash. The hash is a 32-character hexadecimal number.

See also: `crc32()` and `md5_file()`

md5_file (PHP 4 >= 4.2.0)

Calculates the md5 hash of a given filename

string **md5_file** (string filename) \linebreak

Calculates the MD5 hash of the specified *filename* using the RSA Data Security, Inc. MD5 Message-Digest Algorithm (<http://www.faqs.org/rfcs/rfc1321.html>), and returns that hash.

This function has the same purpose of the command line utility md5sum.

See also: md5() and crc32()

metaphone (PHP 4 >= 4.0.0)

Calculate the metaphone key of a string

string **metaphone** (string str) \linebreak

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].

nl_langinfo (PHP 4 >= 4.1.0)

Query language and locale information

string **nl_langinfo** (int item) \linebreak

Warning

This function is currently not documented; only the argument list is available.

nl2br (PHP 3, PHP 4 >= 4.0.0)

Inserts HTML line breaks before all newlines in a string

string **nl2br** (string string) \linebreak

Returns *string* with '
' inserted before all newlines.

Note: Starting with PHP 4.0.5, **nl2br()** is now XHTML compliant. All versions before 4.0.5 will return *string* with '
' inserted before newlines instead of '
'.

See also `htmlspecialchars()`, `htmlspecialchars()` and `wordwrap()`.

ord (PHP 3, PHP 4 >= 4.0.0)

Return ASCII value of character

int **ord** (string string) \linebreak

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";
}
```

You can find an ASCII-table over here: <http://www.mindspring.com/~jc1/serial/Resources/ASCII.html>.

See also `chr()`.

parse_str (PHP 3, PHP 4 >= 4.0.0)

Parses the string into variables

void **parse_str** (string str [, array arr]) \linebreak

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.

Note: Support for the optional second parameter was added in PHP 4.0.3.

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" */
```

See also `set_magic_quotes_runtime()` and `urldecode()`.

print (unknown)

Output a string

print (string arg) \linebreak

Outputs *arg*. Returns TRUE on success, FALSE on failure.

print() is not actually a real function (it is a language construct) so you are not required to use parentheses with it. But **print()**, opposed to `echo()`, can be called using a variable function.

Example 1. print() examples

```
<?php
print("Hello World");

print "print() also works without parentheses.";

print "This spans
multiple lines. The newlines will be
output as well";

print "This spans\nmultiple lines. The newlines will be\noutput as well.";

print "escaping characters is done \"Like this\".";

// You can use variables inside of an print statement
$foo = "foobar";
$bar = "barbaz";

print "foo is $foo"; // foo is foobar

// Using single quotes will print the variable name, not the value
print 'foo is $foo'; // foo is $foo
```

```
// If you are not using any other characters, you can just print variables
print $foo;           // foobar

print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
?>
```

See also `echo()`, `printf()`, and `flush()`.

printf (PHP 3, PHP 4 >= 4.0.0)

Output a formatted string

```
void printf ( string format [, mixed args] ) \linebreak
```

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 >= 4.0.0)

Convert a quoted-printable string to an 8 bit string

```
string quoted_printable_decode ( string str ) \linebreak
```

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 >= 4.0.0)

Quote meta characters

```
string quotemeta ( string str ) \linebreak
```

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 >= 4.0.0)

Strip whitespace from the end of a string

string **rtrim** (string *str* [, string *charlist*] \linebreak

Note: The second parameter was added in PHP 4.1.0

This function returns a string with whitespace stripped from the end of *str*. Without the second parameter, **rtrim()** will strip these characters:

- " " (ASCII 32 (0x20)), an ordinary space.
- "\t" (ASCII 9 (0x09)), a tab.
- "\n" (ASCII 10 (0x0A)), a new line (line feed).
- "\r" (ASCII 13 (0x0D)), a carriage return.
- "\0" (ASCII 0 (0x00)), the NUL-byte.
- "\x0B" (ASCII 11 (0x0B)), a vertical tab.

You can also specify the characters you want to strip, by means of the *charlist* parameter. Simply list all characters that you want to be stripped. With `.` you can specify a range of characters.

Example 1. Usage example of rtrim()

```
<?php

$text = "\t\tThese are a few words :) ... ";
$trimmed = rtrim($text);
// $trimmed = "\t\tThese are a few words :) ..."
$trimmed = rtrim($text, " \t.");
// $trimmed = "\t\tThese are a few words :)"
$clean = rtrim($binary, "\0x00..\0x1F");
// trim the ASCII control characters at the end of $binary
// (from 0 to 31 inclusive)

?>
```

See also `trim()` and `ltrim()`.

setlocale (PHP 3, PHP 4 >= 4.0.0)

Set locale information

string **setlocale** (mixed category, string locale) \linebreak

Category is a named constant (or 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.

Example 1. setlocale() Examples

```
<?php
    /* Set locale to Dutch */
    setlocale (LC_ALL, 'nl_NL');

    /* Output: vrijdag 22 december 1978 */
    echo strftime ("%A %e %B %Y", mktime (0, 0, 0, 12, 22, 1978));
?>
```

similar_text (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Calculate the similarity between two strings

```
int similar_text ( string first, string second [, float percent]) \linebreak
```

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 >= 4.0.0)

Calculate the soundex key of a string

```
string soundex ( string str) \linebreak
```

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';
```

See also levenshtein(), metaphone(), and similar_text().

sprintf (PHP 3, PHP 4 >= 4.0.0)

Return a formatted string

string **sprintf** (string format [, mixed args]) \linebreak

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 float. (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 (signed) decimal number.
 - u - the argument is treated as an integer, and presented as an unsigned decimal number.
 - f - the argument is treated as a float, 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).

As of PHP version 4.0.6 the format string supports argument numbering/swapping. Here is an example:

Example 1. Argument swapping

```
$format = "There are %d monkeys in the %s";
printf($format, $num, $location);
```

This might output, "There are 5 monkeys in the tree". But imagine we are creating a format string in a separate file, commonly because we would like to internationalize it and we rewrite it as:

Example 2. Argument swapping

```
$format = "The %s contains %d monkeys";
printf($format, $num, $location);
```

We now have a problem. The order of the placeholders in the format string does not match the order of the arguments in the code. We would like to leave the code as is and simply indicate in the format string which arguments the placeholders refer to. We would write the format string like this instead:

Example 3. Argument swapping

```
$format = "The %2$s contains %1$d monkeys";
printf($format, $num, $location);
```

An added benefit here is that you can repeat the placeholders without adding more arguments in the code. For example:

Example 4. Argument swapping

```
$format = "The %2$s contains %1$d monkeys.
          That's a nice %2$s full of %1$d monkeys.";
printf($format, $num, $location);
```

See also: `printf()`, `scanf()`, `fscanf()`, and `number_format()`.

Example 5. `sprintf()`: zero-padded integers

```
$isodate = sprintf("%04d-%02d-%02d", $year, $month, $day);
```

Example 6. 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"

```

sscanf (PHP 4)

Parses input from a string according to a format

mixed **sscanf** (string *str*, string *format* [, string *var1*]) \linebreak

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()`.

str_pad (PHP 4)

Pad a string to a certain length with another string

string **str_pad** (string *input*, int *pad_length* [, string *pad_string* [, int *pad_type*]]) \linebreak

This functions returns the *input* string padded 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__"
```

str_repeat (PHP 4 >= 4.0.0)

Repeat a string

string **str_repeat** (string *input*, int *multiplier*) \linebreak

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 "-----".

See also `substr_count()`.

str_replace (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Replace all occurrences of the search string with the replacement string

mixed **str_replace** (mixed search, mixed replace, mixed subject) \linebreak

This function returns a string or an array with all occurrences of *search* in *subject* replaced 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 with 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*.

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()`.

str_rot13 (PHP 4 >= 4.2.0)

Perform the rot13 transform on a string

string **str_rot13** (string str) \linebreak

This function performs the ROT13 encoding on the *str* argument and returns the resulting string. The ROT13 encoding simply shifts every letter by 13 places in the alphabet while leaving non-alpha

characters untouched. Encoding and decoding are done by the same function, passing an encoded string as argument will return the original version.

strcasecmp (PHP 3 >= 3.0.2, PHP 4 >= 4.0.0)

Binary safe case-insensitive string comparison

int **strcasecmp** (string *str1*, string *str2*) \linebreak

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()`, `strcasecmp()`, and `strstr()`.

strchr (PHP 3, PHP 4 >= 4.0.0)

Find the first occurrence of a character

string **strchr** (string *haystack*, string *needle*) \linebreak

This function is an alias for `strstr()`, and is identical in every way.

strcmp (PHP 3, PHP 4 >= 4.0.0)

Binary safe string comparison

int **strcmp** (string *str1*, string *str2*) \linebreak

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()`, `strcasecmp()`, `strcmp()`, and `strstr()`.

strcoll (PHP 4 >= 4.0.5)

Locale based string comparison

int **strcoll** (string str1, string str2) \linebreak

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.

See also `ereg()`, `strcmp()`, `strcasecmp()`, `substr()`, `stristr()`, `strncasecmp()`, `strncmp()`, `strrstr()`, and `setlocale()`.

strcspn (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Find length of initial segment not matching mask

int **strcspn** (string str1, string str2) \linebreak

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 >= 4.0.0)

Strip HTML and PHP tags from a string

string **strip_tags** (string str [, string allowable_tags]) \linebreak

This function tries to return a string with all HTML and PHP tags stripped from a given *str*. 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 and PHP 4.0b3.

Example 1. strip_tags() example

```
$string = strip_tags($string, '<a><b><i><u>');
```

Warning

This function does not modify any attributes on the tags that you allow using *allowable_tags*, including the *style* and *onmouseover* attributes that a mischievous user may abuse when posting text that will be shown to other users.

stripslashes (PHP 4 >= 4.0.0)

Un-quote string quoted with addslashes()

string **stripslashes** (string str) \linebreak

Returns a string with backslashes stripped off. Recognizes C-like \n, \r ..., octal and hexadecimal representation.

See also addslashes().

stripslashes (PHP 3, PHP 4 >= 4.0.0)

Un-quote string quoted with addslashes()

string **stripslashes** (string str) \linebreak

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 >= 4.0.0)

Case-insensitive strstr()

string **stristr** (string haystack, string needle) \linebreak

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(), strtolower(), substr(), and ereg().

strlen (PHP 3, PHP 4 >= 4.0.0)

Get string length

```
int strlen ( string str) \linebreak
```

Returns the length of *string*.

strnatcasecmp (PHP 4 >= 4.0.0)

Case insensitive string comparisons using a "natural order" algorithm

```
int strnatcasecmp ( string str1, string str2) \linebreak
```

This function implements a comparison algorithm that orders alphanumeric strings in the way a human being would. The behaviour 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://naturalordersort.org/>) 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 `strstr()`.

strnatcmp (PHP 4 >= 4.0.0)

String comparisons using a "natural order" algorithm

```
int strnatcmp ( string str1, string str2) \linebreak
```

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://naturalordersort.org/>) 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()`, `strstr()`, `natsort()` and `natcasesort()`.

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) \linebreak
```

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 `strstr()`.

strncmp (PHP 4 >= 4.0.0)

Binary safe string comparison of the first n characters

`int strncmp (string str1, string str2, int len) \linebreak`

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()`, `strncasecmp()`, `strcasecmp()`, `substr()`, `stristr()`, `strcmp()`, and `strstr()`.

strpos (PHP 3, PHP 4 \geq 4.0.0)

Find position of first occurrence of a string

`int strpos (string haystack, string needle [, int offset]) \linebreak`

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_integer($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()`, `strchr()`, `substr()`, `stristr()`, and `strstr()`.

strrchr (PHP 3, PHP 4 >= 4.0.0)

Find the last occurrence of a character in a string

string **strrchr** (string haystack, string needle) \linebreak

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 `strchr()`, `substr()`, `stristr()`, and `strstr()`.

strrev (PHP 3, PHP 4 >= 4.0.0)

Reverse a string

string **strrev** (string string) \linebreak

Returns *string*, reversed.

Example 1. Reversing a string with strrev()

```
<php
echo strrev("Hello world!"); // outputs "!dlrow olleH"
?>
```

strrpos (PHP 3, PHP 4 >= 4.0.0)

Find position of last occurrence of a char in a string

```
int strrpos ( string haystack, char needle) \linebreak
```

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()`, `strchr()`, `substr()`, `stristr()`, and `strstr()`.

strspn (PHP 3>= 3.0.3, PHP 4 >= 4.0.0)

Find length of initial segment matching mask

```
int strspn ( string str1, string str2) \linebreak
```

Returns the length of the initial segment of *str1* which consists entirely of characters in *str2*.

The line of code:

```
$var = strspn("42 is the answer, what is the question ...", "1234567890");
```

will assign 2 to `$var`, because the string "42" will be the longest segment containing characters from "1234567890".

See also `strcspn()`.

strstr (PHP 3, PHP 4 >= 4.0.0)

Find first occurrence of a string

string **strstr** (string haystack, string needle) \linebreak

Returns part of *haystack* string from the first occurrence of *needle* to the end of *haystack*.

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: This function is case-sensitive. For case-insensitive searches, use `stristr()`.

Example 1. strstr() example

```
$email = 'user@example.com';
$domain = strstr($email, '@');
print $domain; // prints @example.com
```

See also `ereg()`, `preg_match()`, `strchr()`, `stristr()`, `strpos()`, `strrchr()`, and `substr()`.

strtok (PHP 3, PHP 4 >= 4.0.0)

Tokenize string

string **strtok** (string arg1, string arg2) \linebreak

strtok() splits a string (*arg1*) into smaller strings (tokens), with each token being delimited by any character from *arg2*. 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\tan example\nstring";
/* Use tab and newline as tokenizing characters as well */
$tok = strtok($string, " \n\t");
while ($tok) {
    echo "Word=$tok<br>";
    $tok = strtok(" \n\t");
}

```

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.

The behavior when an empty part was found changed with PHP 4.1.0. The old behavior returned an empty string, while the new, correct, behavior simply skips the part of the string:

Example 2. Old strtok() behavior

```

$first_token = strtok('/something', '/');
$second_token = strtok('/');
var_dump ($first_token, $second_token);

/* Output:
string(0) ""
string(9) "something"
*/

```

Example 3. New strtok() behavior

```

$first_token = strtok('/something', '/');
$second_token = strtok('/');
var_dump ($first_token, $second_token);

/* Output:
string(9) "something"
bool(false)
*/

```

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 >= 4.0.0)

Make a string lowercase

string **strtolower** (string str) \linebreak

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()`, `ucfirst()`, and `ucwords()`.

strtoupper (PHP 3, PHP 4 >= 4.0.0)

Make a string uppercase

string **strtoupper** (string string) \linebreak

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()`, `ucfirst()`, and `ucwords()`.

strtr (PHP 3, PHP 4 >= 4.0.0)

Translate certain characters

string **strtr** (string *str*, string *from*, string *to*) \linebreak string **strtr** (string *str*, array *replace_pairs*) \linebreak

This function returns a copy of *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 optional *to* and *from* parameters were added in PHP 4.0.0

See also `ereg_replace()`.

substr (PHP 3, PHP 4 >= 4.0.0)

Return part of a string

string **substr** (string *string*, int *start* [, int *length*]) \linebreak

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.

Example 1. Basic substr() usage

```
$rest = substr("abcdef", 1); // returns "bcdef"
$rest = substr("abcdef", 1, 3); // returns "bcd"
$rest = substr("abcdef", 0, 4); // returns "abcd"
$rest = substr("abcdef", 0, 8); // returns "abcdef"
```

If *start* is negative, the returned string will start at the *start*'th character from the end of *string*.

Example 2. Using a negative *start*

```
$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 contain at most *length* characters beginning from *start* (depending on the length of *string*. If *string* is less than *start* characters long, FALSE will be returned.

If *length* is given and is negative, then that many characters will be omitted from the end of *string* (after the start position has been calculated when a *start* is negative). If *start* denotes a position beyond this truncation, an empty string will be returned.

Example 3. Using a negative *length*

```
$rest = substr("abcdef", 0, -1); // returns "abcde"
$rest = substr("abcdef", 2, -1); // returns "cde"
$rest = substr("abcdef", 4, -4); // returns ""
$rest = substr("abcdef", -3, -1); // returns "de"
```

See also strchr() and ereg().

substr_count (PHP 4 >= 4.0.0)

Count the number of substring occurrences

int **substr_count** (string haystack, string needle) \linebreak

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 >= 4.0.0)

Replace text within a portion of a string

string **substr_replace** (string string, string replacement, int start [, int length]) \linebreak

substr_replace() replaces a copy 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<hr>\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()`.

trim (PHP 3, PHP 4 >= 4.0.0)

Strip whitespace from the beginning and end of a string

string **trim** (string *str* [, string *charlist*] \linebreak

Note: The optional *charlist* parameter was added in PHP 4.1.0

This function returns a string with whitespace stripped from the beginning and end of *str*. Without the second parameter, **trim()** will strip these characters:

- " " (ASCII 32 (0x20)), an ordinary space.
- "\t" (ASCII 9 (0x09)), a tab.
- "\n" (ASCII 10 (0x0A)), a new line (line feed).
- "\r" (ASCII 13 (0x0D)), a carriage return.
- "\0" (ASCII 0 (0x00)), the NUL-byte.
- "\x0B" (ASCII 11 (0x0B)), a vertical tab.

You can also specify the characters you want to strip, by means of the *charlist* parameter. Simply list all characters that you want to be stripped. With `.` you can specify a range of characters.

Example 1. Usage example of trim()

```
<?php
$text = "\t\tThese are a few words :) ... ";
$trimmed = trim($text);
// $trimmed = "These are a few words :) ..."
$trimmed = trim($text, " \t.");
// $trimmed = "These are a few words :)"
$clean = trim($binary, "\0x00..\0x1F");
// trim the ASCII control characters at the beginning and end of $binary
// (from 0 to 31 inclusive)
```

?>

See also `ltrim()` and `rtrim()`.

ucfirst (PHP 3, PHP 4 >= 4.0.0)

Make a string's first character uppercase

string **ucfirst** (string *str*) \linebreak

Returns a string with the first character of *str* capitalized, 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

```
$foo = 'hello world!';
$foo = ucfirst($foo);           // Hello world!

$bar = 'HELLO WORLD!';
$bar = ucfirst($bar);          // HELLO WORLD!
$bar = ucfirst(strtolower($bar)); // Hello world!
```

See also `strtolower()`, `strtoupper()`, and `ucwords()`.

ucwords (PHP 3 >= 3.0.3, PHP 4 >= 4.0.0)

Uppercase the first character of each word in a string

string **ucwords** (string *str*) \linebreak

Returns a string with the first character of each word in *str* capitalized, if that character is alphabetic.

Example 1. ucwords() example

```
$foo = 'hello world!';
$foo = ucwords($foo);           // Hello World!

$bar = 'HELLO WORLD!';
$bar = ucwords($bar);           // HELLO WORLD!
$bar = ucwords(strtolower($bar)); // Hello World!
```

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()`.

vprintf (PHP 4 >= 4.1.0)

Output a formatted string

```
void vprintf ( string format, array args) \linebreak
```

Display array values as a formatted string according to *format* (which is described in the documentation for `sprintf()`).

Operates as `printf()` but accepts an array of arguments, rather than a variable number of arguments.

See also: `printf()`, `sprintf()`, `vsprintf()`

vsprintf (PHP 4 >= 4.1.0)

Return a formatted string

```
string vsprintf ( string format, array args) \linebreak
```

Return array values as a formatted string according to *format* (which is described in the documentation for `sprintf()`).

Operates as `sprintf()` but accepts an array of arguments, rather than a variable number of arguments.

See also: `sprintf()`, **`vsprintf()`**, `vprintf()`

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*]]) \linebreak

Returns a string with *str* wrapped 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 apart. (See second example).

Note: The optional *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()`.

XCVII. Sybase functions

sybase_affected_rows (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

get number of affected rows in last query

```
int sybase_affected_rows ( [int link_identifier] ) \linebreak
```

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 >= 4.0.0)

close Sybase connection

```
bool sybase_close ( int link_identifier ) \linebreak
```

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 >= 4.0.0)

open Sybase server connection

```
int sybase_connect ( string servername, string username, string password [, string charset] ) \linebreak
```

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 >= 4.0.0)

move internal row pointer

bool **sybase_data_seek** (int result_identifier, int row_number) \linebreak

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 >= 4.0.0)

fetch row as array

array **sybase_fetch_array** (int result) \linebreak

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 >= 4.0.0)

get field information

object **sybase_fetch_field** (int result [, int field_offset]) \linebreak

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 >= 4.0.0)

fetch row as object

int **sybase_fetch_object** (int result) \linebreak

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 >= 4.0.0)

get row as enumerated array

array **sybase_fetch_row** (int result) \linebreak

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_row()** 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 >= 4.0.0)

set field offset

```
int sybase_field_seek ( int result, int field_offset) \linebreak
```

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 >= 4.0.0)

free result memory

```
bool sybase_free_result ( int result) \linebreak
```

`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 >= 4.0.0)

Returns the last message from the server

```
string sybase_get_last_message ( void) \linebreak
```

`sybase_get_last_message()` returns the last message reported by the server.

sybase_min_client_severity (PHP 3, PHP 4 >= 4.0.0)

Sets minimum client severity

```
void sybase_min_client_severity ( int severity) \linebreak
```

`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 >= 4.0.0)

Sets minimum error severity

```
void sybase_min_error_severity ( int severity) \linebreak
```

sybase_min_error_severity() sets the minimum error severity level.

See also: `sybase_min_message_severity()`.

sybase_min_message_severity (PHP 3, PHP 4 >= 4.0.0)

Sets minimum message severity

```
void sybase_min_message_severity ( int severity) \linebreak
```

sybase_min_message_severity() sets the minimum message severity level.

See also: `sybase_min_error_severity()`.

sybase_min_server_severity (PHP 3, PHP 4 >= 4.0.0)

Sets minimum server severity

```
void sybase_min_server_severity ( int severity) \linebreak
```

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 >= 4.0.0)

get number of fields in result

```
int sybase_num_fields ( int result) \linebreak
```

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 >= 4.0.0)

get number of rows in result

```
int sybase_num_rows ( int result) \linebreak
```

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 >= 4.0.0)

open persistent Sybase connection

```
int sybase_pconnect ( string servername, string username, string password [, string charset]) \linebreak
```

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 >= 4.0.0)

send Sybase query

```
int sybase_query ( string query, int link_identifier) \linebreak
```

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 >= 4.0.0)

get result data

string **sybase_result** (int result, int row, mixed field) \linebreak

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 >= 4.0.0)

select Sybase database

bool **sybase_select_db** (string database_name, int link_identifier) \linebreak

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()

XCVIII. URL Functions

base64_decode (PHP 3, PHP 4 >= 4.0.0)

Decodes data encoded with MIME base64

string **base64_decode** (string *encoded_data*) \linebreak

base64_decode() decodes *encoded_data* and returns the original data. The returned data may be binary.

See also: `base64_encode()`, RFC2045 section 6.8.

base64_encode (PHP 3, PHP 4 >= 4.0.0)

Encodes data with MIME base64

string **base64_encode** (string *data*) \linebreak

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()`, RFC2045 section 6.8.

parse_url (PHP 3, PHP 4 >= 4.0.0)

Parse a URL and return its components

array **parse_url** (string *url*) \linebreak

This function returns an associative array returning any of the various components of the URL that are present. This includes the

- *scheme* - e.g. `http://`
- *host*
- *port*
- *user*
- *pass*
- *path*
- *query* - after the question mark `?`
- *fragment* - after the hashmark `#`

See also `pathinfo()`.

rawurldecode (PHP 3, PHP 4 >= 4.0.0)

Decode URL-encoded strings

string **rawurldecode** (string str) \linebreak

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
```

.

Note: **rawurldecode()** does not decode plus symbols ('+') into spaces. **urldecode()** does.

See also **rawurlencode()**, **urldecode()**, **urlencode()**.

rawurlencode (PHP 3, PHP 4 >= 4.0.0)

URL-encode according to RFC1738

string **rawurlencode** (string str) \linebreak

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()`.

urldecode (PHP 3, PHP 4 >= 4.0.0)

Decodes URL-encoded string

string **urldecode** (string str) \linebreak

Decodes any %## encoding in the given string. The decoded string is returned.

Example 1. urldecode() example

```
$a = explode('&', $QUERY_STRING);
$i = 0;
while ($i < count($a)) {
    $b = split('=', $a[$i]);
    echo 'Value for parameter ', htmlspecialchars(urldecode($b[0])),
        ' is ', htmlspecialchars(urldecode($b[1])), "<br />\n";
    $i++;
}
```

See also `urlencode()`, `rawurlencode()`, `rawurldecode()`.

urlencode (PHP 3, PHP 4 >= 4.0.0)

URL-encodes string

string **urlencode** (string str) \linebreak

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()`.

XCIX. Variable Functions

For information on how variables behave, see the Variables entry in the Language Reference section of the manual.

doubleval (PHP 3, PHP 4 >= 4.0.0)

Alias of floatval()

This function is an alias of floatval().

Note: This alias is a left-over from a function-renaming. In older versions of PHP you'll need to use this alias of the floatval() function, because floatval() wasn't yet available in that version.

empty (unknown)

Determine whether a variable is set

boolean **empty** (mixed var) \linebreak

Note: empty() is a language construct.

This is the opposite of (boolean) *var*, except that no warning is generated when the variable is not set. See converting to boolean for more information.

```
$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()`.

floatval (PHP 4 >= 4.2.0)

Get float value of a variable

float **floatval** (mixed var) \linebreak

Returns the float value of *var*.

Var may be any scalar type. You cannot use **floatval()** on arrays or objects.

```
$var = '122.34343The';
$float_value_of_var = floatval ($var);
print $float_value_of_var; // prints 122.34343
```

See also `intval()`, `strval()`, `settype()` and Type juggling.

get_defined_vars (PHP 4 >= 4.0.4)

Returns an array of all defined variables

array **get_defined_vars** (void) \linebreak

This function returns an multidimensional array containing a list of all defined variables, be them environment, server or user-defined variables.

```
$b = array(1,1,2,3,5,8);

$arr = get_defined_vars();

// print $b
print_r($arr["b"]);

// print path to the PHP interpreter (if used as a CGI)
// e.g. /usr/local/bin/php
echo $arr["_"];

// print the command-line paramaters if any
print_r($arr["argv"]);

// print all the server vars
print_r($arr["HTTP_SERVER_VARS"]);

// print all the available keys for the arrays of variables
print_r(array_keys(get_defined_vars()));
```


See also `get_defined_functions()` and `get_defined_constants()`.

get_resource_type (PHP 4 >= 4.0.2)

Returns the resource type

string **get_resource_type** (resource handle) \linebreak

This function returns a string representing the type of the resource passed to it. If the parameter is not a valid resource, it generates an error.

```
$c = mysql_connect();
echo get_resource_type($c)."\n";
// prints: mysql link

$fp = fopen("foo","w");
echo get_resource_type($fp)."\n";
// prints: file

$doc = new_xmldoc("1.0");
echo get_resource_type($doc->doc)."\n";
// prints: domxml document
```

gettype (PHP 3, PHP 4 >= 4.0.0)

Get the type of a variable

string **gettype** (mixed var) \linebreak

Returns the type of the PHP variable *var*.

Warning

Never use **gettype()** to test for a certain type, since the returned string may be subject to change in a future version. In addition, it is slow too, as it involves string comparison .

Instead, use the `is_*` functions.

Possible values for the returned string are:

- "boolean" (since PHP 4)
- "integer"
- "double" (for historical reasons "double" is returned in case of a float, and not simply "float")
- "string"
- "array"
- "object"
- "resource" (since PHP 4)
- "NULL" (since PHP 4)
- "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()`, `is_array()`, `is_bool()`, `is_float()`, `is_integer()`, `is_null()`, `is_numeric()`, `is_object()`, `is_resource()`, `is_scalar()`, and `is_string()`.

import_request_variables (PHP 4 >= 4.1.0)

Import GET/POST/Cookie variables into the global scope

```
bool import_request_variables ( string types [, string prefix] ) \linebreak
```

Imports GET/POST/Cookie variables into the global scope. It is useful if you disabled `register_globals`, but would like to see some variables in the global scope.

Using the *types* parameter, you can specify, which request variables to import. You can use 'G', 'P' and 'C' characters respectively for GET, POST and Cookie. These characters are not case sensitive, so you can also use any combination of 'g', 'p' and 'c'. POST includes the uploaded file informations. Note, that the order of the letters matters, as using "gp", the POST variables will overwrite GET variables with the same name. Any other other letters then GPC are discarded.

Note: Although the *prefix* argument is optional, you will get a notice level error, if you specify no prefix, or specify an empty string as a prefix. This is a possible security hazard. Notice level errors are not displayed using the default error reporting level.

```
// This will import GET and POST vars
```

```
// with an "rvar_" prefix
import_request_variables("gP", "rvar_");
```

See also `register_globals` and `track_vars`.

intval (PHP 3, PHP 4 >= 4.0.0)

Get integer value of a variable

```
int intval ( mixed var [, int base]) \linebreak
```

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.

Note: The *base* argument for **intval()** has no effect unless the *var* argument is a string.

See also `floatval()`, `strval()`, `settype()` and Type juggling.

is_array (PHP 3, PHP 4 >= 4.0.0)

Finds whether a variable is an array

```
bool is_array ( mixed var) \linebreak
```

Returns `TRUE` if *var* is an array, `FALSE` otherwise.

See also `is_float()`, `is_int()`, `is_integer()`, `is_string()`, and `is_object()`.

is_bool (PHP 4 >= 4.0.0)

Finds out whether a variable is a boolean

```
bool is_bool ( mixed var) \linebreak
```

Returns `TRUE` if the *var* parameter is a boolean.

See also `is_array()`, `is_float()`, `is_int()`, `is_integer()`, `is_string()`, and `is_object()`.

is_callable (PHP 4 >= 4.0.6)

Find out whether the argument is a valid callable construct

```
bool is_callable ( mixed var [, bool syntax_only [, string callable_name]]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

is_double (PHP 3, PHP 4 >= 4.0.0)

Alias of is_float()

This function is an alias of is_float().

is_float (PHP 3, PHP 4 >= 4.0.0)

Finds whether a variable is a float

```
bool is_float ( mixed var) \linebreak
```

Returns `TRUE` if `var` is a float, `FALSE` otherwise.

Note: To test if a variable is a number or a numeric string (such as form input, which is always a string), you must use `is_numeric()`.

See also `is_bool()`, `is_int()`, `is_integer()`, `is_numeric()`, `is_string()`, `is_array()`, and `is_object()`,

is_int (PHP 3, PHP 4 >= 4.0.0)

Find whether a variable is an integer

```
bool is_int ( mixed var) \linebreak
```

Returns `TRUE` if `var` is an integer `FALSE` otherwise.

Note: To test if a variable is a number or a numeric string (such as form input, which is always a string), you must use `is_numeric()`.

See also `is_bool()`, `is_float()`, `is_integer()`, `is_numeric()`, `is_string()`, `is_array()`, and `is_object()`.

is_integer (PHP 3, PHP 4 >= 4.0.0)

Alias of `is_int()`

This function is an alias of `is_int()`.

is_long (PHP 3, PHP 4 >= 4.0.0)

Alias of `is_int()`

This function is an alias of `is_int()`.

is_null (PHP 4 >= 4.0.4)

Finds whether a variable is `NULL`

```
bool is_null ( mixed var) \linebreak
```

Returns `TRUE` if `var` is null, `FALSE` otherwise.

See also `is_bool()`, `is_numeric()`, `is_float()`, `is_int()`, `is_string()`, `is_object()`, `is_array()`, `is_integer()`, and `is_real()`

is_numeric (PHP 4 >= 4.0.0)

Finds whether a variable is a number or a numeric string

```
bool is_numeric ( mixed var) \linebreak
```

Returns `TRUE` if `var` is a number or a numeric string, `FALSE` otherwise.

See also `is_bool()`, `is_float()`, `is_int()`, `is_string()`, `is_object()`, `is_array()`, and `is_integer()`.

is_object (PHP 3, PHP 4 >= 4.0.0)

Finds whether a variable is an object

```
bool is_object ( mixed var) \linebreak
```

Returns `TRUE` if `var` is an object, `FALSE` otherwise.

See also `is_bool()`, `is_int()`, `is_integer()`, `is_float()`, `is_string()`, and `is_array()`.

is_real (PHP 3, PHP 4 >= 4.0.0)

Alias of `is_float()`

This function is an alias of `is_float()`.

is_resource (PHP 4 >= 4.0.0)

Finds whether a variable is a resource

```
bool is_resource ( mixed var) \linebreak
```

`is_resource()` returns `TRUE` if the variable given by the `var` parameter is a resource, otherwise it returns `FALSE`.

See the documentation on the resource-type for more information.

is_scalar (PHP 4 >= 4.0.5)

Finds whether a variable is a scalar

```
bool is_scalar ( mixed var) \linebreak
```

`is_scalar()` returns `TRUE` if the variable given by the `var` parameter is a scalar, otherwise it returns `FALSE`.

Scalar variables are those containing an integer, float, string or boolean. Types array, object and resource or not scalar.

```
function show_var($var) {  
    if (is_scalar($var)) {  
        echo $var;  
    } else {
```

```

        var_dump($var);
    }
}
$pi = 3.1416;
$proteins = array("hemoglobin", "cytochrome c oxidase", "ferredoxin");

show_var($pi);
// prints: 3.1416

show_var($proteins)
// prints:
// array(3) {
//   [0]=>
//   string(10) "hemoglobin"
//   [1]=>
//   string(20) "cytochrome c oxidase"
//   [2]=>
//   string(10) "ferredoxin"
// }

```

Note: `is_scalar()` does not consider resource type values to be scalar as resources are abstract datatypes which are currently based on integers. This implementation detail should not be relied upon, as it may change.

See also `is_bool()`, `is_numeric()`, `is_float()`, `is_int()`, `is_real()`, `is_string()`, `is_object()`, `is_array()`, and `is_integer()`.

is_string (PHP 3, PHP 4 >= 4.0.0)

Finds whether a variable is a string

bool **is_string** (mixed var) \linebreak

Returns `TRUE` if `var` is a string, `FALSE` otherwise.

See also `is_bool()`, `is_int()`, `is_integer()`, `is_float()`, `is_real()`, `is_object()`, and `is_array()`.

isset (unknown)

Determine whether a variable is set

boolean **isset** (mixed var [, mixed var [, ...]]) \linebreak

Note: `isset()` is a language construct.

Returns `TRUE` if `var` exists; `FALSE` otherwise.

If a variable has been unset with `unset()`, it will no longer be `isset()`. `isset()` will return `FALSE` if testing a variable that has been set to `NULL`. Also note that a `NULL` byte ("`\0`") is not equivalent to the PHP `NULL` constant.

```
$a = "test";
$b = "anothertest";

echo isset ($a); // TRUE
echo isset ($a, $b) //TRUE

unset ($a);
echo isset ($a); // FALSE
echo isset ($a, $b); //FALSE

$foo = NULL;
print isset ($foo); // FALSE
```

See also `empty()` and `unset()`.

print_r (PHP 4 >= 4.0.0)

Prints human-readable information about a variable

void **print_r** (mixed expression) \linebreak

print_r() displays information about a variable in a way that's readable by humans. If given a string, integer or float, 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.

Remember that **print_r()** will move the array pointer to the end. Use `reset()` to bring it back to beginning.

Tip: As with anything that outputs its result directly to the browser, you can use the output-control functions to capture the output of this function, and save it - for example - in a string.

<pre>


```
<?php
    $a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x','y','z'));
    print_r ($a);
?>
</pre>
```

Which will output:

```
<pre>
Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

Note: Prior to PHP 4.0.4, **print_r()** will continue forever if given an array or object that contains a direct or indirect reference to itself. An example is `print_r($GLOBALS)` because `$GLOBALS` is itself a global variable that contains a reference to itself.

See also `ob_start()`, `var_dump()`, and `var_export()`.

serialize (PHP 3>= 3.0.5, PHP 4 >= 4.0.0)

Generates a storable representation of a value

string **serialize** (mixed value) \linebreak

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 all types, except the resource-type. You can even **`serialize()`** arrays that contain references to itself. References inside the array/object you are **`serialize()`**ing will also be stored.

Note: In PHP 3, object properties will be serialized, but methods are lost. PHP 4 removes that limitation and restores both properties and methods. Please see the Serializing Objects section of Classes and Objects for more information.

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. */
    }
}
```

See Also: `unserialize()`.

settype (PHP 3, PHP 4 >= 4.0.0)

Set the type of a variable

bool settype (mixed var, string type) \linebreak

Set the type of variable *var* to *type*.

Possible values of *type* are:

- "boolean" (or, since PHP 4.2.0, "bool")
- "integer" (or, since PHP 4.2.0, "int")
- "float" (only possible since PHP 4.2.0, for older versions use the deprecated variant "double")

- "string"
- "array"
- "object"
- "null" (since PHP 4.0.8)

Returns `TRUE` if successful; otherwise returns `FALSE`.

Example 1. `settype()` example

```
$foo = "5bar"; // string
$bar = true;  // boolean

settype($foo, "integer"); // $foo is now 5 (integer)
settype($bar, "string");  // $bar is now "1" (string)
```

See also `gettype()`, type-casting and type-juggling.

strval (PHP 3, PHP 4 >= 4.0.0)

Get string value of a variable

string **strval** (mixed var) \linebreak

Returns the string value of *var*. See the documentation on `string` for more information on converting to string.

var may be any scalar type. You cannot use **strval()** on arrays or objects.

See also `floatval()`, `intval()`, `settype()` and Type juggling.

unserialize (PHP 3 >= 3.0.5, PHP 4 >= 4.0.0)

Creates a PHP value from a stored representation

mixed **unserialize** (string str) \linebreak

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, float, string, array or object.

Note: It's possible to set a callback-function which will be called, if an undefined class should be instantiated during unserializing. (to prevent getting an incomplete object "__PHP_Incomplete_Class".) Use your `php.ini`, `ini_set()` or `.htaccess`-file to define 'unserialize_callback_func'. Everytime an undefined class should be instantiated, it'll be called. To disable this feature just empty this setting.

Example 1. unserialize_callback_func example

```
$serialized_object='O:1:"a":1:{s:5:"value";s:3:"100"}';

ini_set('unserialize_callback_func','mycallback'); // set your callback_function

function mycallback($classname) {
    // just include a file containing your classdefinition
    // you get $classname to figure out which classdefinition is required
}
```

Note: In PHP 3, methods are not preserved when unserializing a serialized object. PHP 4 removes that limitation and restores both properties and methods. Please see the Serializing Objects section of Classes and Objects or more information.

Example 2. unserialize() example

```
// Here, we use unserialize() to load session data to the
// $session_data array from the string selected from a database.
// 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();
    }
}
```

See Also: `serialize()`.

unset (unknown)

Unset a given variable

```
void unset ( mixed var [, mixed var [, ...]]) \linebreak
```

Note: `unset()` is a language construct.

`unset()` destroys the specified variables. Note that in PHP 3, `unset()` will always return `TRUE` (actually, the integer value 1). In PHP 4, however, `unset()` is no longer a true function: it is now a statement. As such no value is returned, and attempting to take the value of `unset()` results in a parse error.

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()** destroys the variable and all its references.

```
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 >= 4.0.0)

Dumps information about a variable

```
void var_dump ( mixed expression [, mixed expression [, ...]] ) \linebreak
```

This function returns structured information about one or more expressions that includes its type and value. Arrays are explored recursively with values indented to show structure.

Tip: As with anything that outputs its result directly to the browser, you can use the output-control functions to capture the output of this function, and save it - for example - in a string.

Compare **var_dump()** to `print_r()`.

```

<pre>
<?php
$a = array (1, 2, array ("a", "b", "c"));
var_dump ($a);

/* output:
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  array(3) {
    [0]=>
    string(1) "a"
    [1]=>
    string(1) "b"
    [2]=>
    string(1) "c"
  }
}
*/

$b = 3.1;
$c = TRUE;
var_dump($b,$c);

/* output:
float(3.1)
bool(true)

*/
?>
</pre>

```

var_export (PHP 4 >= 4.2.0)

Outputs or returns a string representation of a variable

mixed **var_export** (mixed expression [, bool return]) \linebreak

This function returns structured information about the variable that is passed to this function. It is similar to `var_dump()` with the exception that the returned representation is valid PHP code.

You can also return the variable representation by using `TRUE` as second parameter to this function.

Compare `var_export()` to `var_dump()`.

```
<pre>
<?php
$a = array (1, 2, array ("a", "b", "c"));
var_export ($a);

/* output:
array (
  0 => 1,
  1 => 2,
  2 =>
  array (
    0 => 'a',
    1 => 'b',
    2 => 'c',
  ),
)
*/

$b = 3.1;
$v = var_export($b, TRUE);
echo $v;

/* output:
3.1
*/
?>
</pre>
```

C. vpopmail functions

vpopmail_add_alias_domain (PHP 4 >= 4.0.5)

Add an alias for a virtual domain

```
bool vpopmail_add_alias_domain ( string domain, string aliasdomain) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_add_alias_domain_ex (PHP 4 >= 4.0.5)

Add alias to an existing virtual domain

```
bool vpopmail_add_alias_domain_ex ( string olddomain, string newdomain) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_add_domain (PHP 4 >= 4.0.5)

Add a new virtual domain

```
bool vpopmail_add_domain ( string domain, string dir, int uid, int gid) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_add_domain_ex (PHP 4 >= 4.0.5)

Add a new virtual domain

bool **vpopmail_add_domain_ex** (string domain, string passwd [, string quota [, string bounce [, bool apop]])
\linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_add_user (PHP 4 >= 4.0.5)

Add a new user to the specified virtual domain

bool **vpopmail_add_user** (string user, string domain, string password [, string gecost [, bool apop]]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_alias_add (PHP 4 >= 4.1.0)

insert a virtual alias

bool **vpopmail_alias_add** (string user, string domain, string alias) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_alias_del (PHP 4 >= 4.1.0)

deletes all virtual aliases of a user

bool **vpopmail_alias_del** (string user, string domain) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_alias_del_domain (PHP 4 >= 4.1.0)

deletes all virtual aliases of a domain

bool **vpopmail_alias_del_domain** (string domain) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_alias_get (PHP 4 >= 4.1.0)

get all lines of an alias for a domain

array **vpopmail_alias_get** (string alias, string domain) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_alias_get_all (PHP 4 >= 4.1.0)

get all lines of an alias for a domain

array **vpopmail_alias_get_all** (string domain) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_auth_user (PHP 4 >= 4.0.5)

Attempt to validate a username/domain/password. Returns true/false

bool **vpopmail_auth_user** (string user, string domain, string password [, string apop]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_del_domain (PHP 4 >= 4.0.5)

Delete a virtual domain

```
bool vpopmail_del_domain ( string domain) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_del_domain_ex (PHP 4 >= 4.0.5)

Delete a virtual domain

```
bool vpopmail_del_domain_ex ( string domain) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_del_user (PHP 4 >= 4.0.5)

Delete a user from a virtual domain

```
bool vpopmail_del_user ( string user, string domain) \linebreak
```


Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_error (PHP 4 >= 4.0.5)

Get text message for last vpopmail error. Returns string

string **vpopmail_error** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_passwd (PHP 4 >= 4.0.5)

Change a virtual user's password

bool **vpopmail_passwd** (string user, string domain, string password) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

vpopmail_set_user_quota (PHP 4 >= 4.0.5)

Sets a virtual user's quota

bool **vpopmail_set_user_quota** (string user, string domain, string quota) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

CI. W32api functions

w32api_deftype (PHP 4 >= 4.2.0)

Defines a type for use with other w32api_functions

int **w32api_deftype** (string typename, string member1_type, string member1_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

w32api_init_dtype (PHP 4 >= 4.2.0)

Creates an instance to the data type typename and fills it with the values val1, val2, the function then returns a DYNAPARM which can be passed when invoking a function as a parameter

resource **w32api_init_dtype** (string typename, mixed val1, mixed val2) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

w32api_invoke_function (unknown)

Invokes function funcname with the arguments passed after the function name

mixed **w32api_invoke_function** (string funcname) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

w32api_register_function (PHP 4 >= 4.2.0)

Registers function function_name from library with PHP

bool **w32api_register_function** (string library, string function_name) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

w32api_set_call_method (PHP 4 >= 4.2.0)

Sets the calling method used

void **w32api_set_call_method** (int method) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

CII. WDDX Functions

These functions are intended for work with WDDX (<http://www.openwddx.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: If you want to serialize non-ASCII characters you have to set the appropriate locale before doing so (see `setlocale()`).

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_add_vars (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Add variables to a WDDX packet with the specified ID

wddx_add_vars (int packet_id, mixed name_var [, mixed ...]) \linebreak

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 >= 4.0.0)

Deserializes a WDDX packet

mixed **wddx_deserialize** (string packet) \linebreak

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.

wddx_packet_end (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Ends a WDDX packet with the specified ID

string **wddx_packet_end** (int packet_id) \linebreak

wddx_packet_end() ends the WDDX packet specified by the *packet_id* and returns the string with the packet.

wddx_packet_start (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Starts a new WDDX packet with structure inside it

int **wddx_packet_start** ([string comment]) \linebreak

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_serialize_value (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Serialize a single value into a WDDX packet

string **wddx_serialize_value** (mixed var [, string comment]) \linebreak

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 >= 4.0.0)

Serialize variables into a WDDX packet

string **wddx_serialize_vars** (mixed var_name [, mixed ...]) \linebreak

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";

$clvars = array("c", "d");
print wddx_serialize_vars("a", "b", $clvars);
?>
```

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>
```

CIII. 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://sourceforge.net/projects/expat/>.

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 `LDFLAGS` 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. |
| <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 structure 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

```

<?php
$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, $attribs) {
    print "&lt;<font color=\"#0000cc\">$name</font>";
    if (sizeof($attribs)) {
        while (list($k, $v) = each($attribs)) {
            print " <font color=\"#009900\">$k</font>=&lt;<font
                color=\"#990000\">$v</font>&lt;&lt;";
        }
    }
    print "&gt;";
}

function endElement($parser, $name) {
    print "&lt;/&lt;font color=\"#0000cc\">$name</font>&gt;";
}

function characterData($parser, $data) {
    print "&lt;&lt;b>$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;
<section id="about">
  <title>About this Document</title>
  <para>
    <!-- this is a comment -->
    <?php print 'Hi! This is PHP version '.phpversion(); ?>
  </para>
</section>
</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>

```

utf8_decode (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Converts a string with ISO-8859-1 characters encoded with UTF-8 to single-byte ISO-8859-1.

string **utf8_decode** (string *data*) \linebreak

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 >= 4.0.0)

encodes an ISO-8859-1 string to UTF-8

string **utf8_encode** (string *data*) \linebreak

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 | 0bbbbbbb |
| 2 | 11 | 110bbbb 10bbbbbb |
| 3 | 16 | 1110bbbb 10bbbbbb 10bbbbbb |
| 4 | 21 | 11110bbb 10bbbbbb 10bbbbbb 10bbbbbb |

Each *b* represents a bit that can be used to store character data.

xml_error_string (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

get XML parser error string

string **xml_error_string** (int *code*) \linebreak

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_byte_index (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

get current byte index for an XML parser

```
int xml_get_current_byte_index ( resource parser) \linebreak
```

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_get_current_column_number (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

Get current column number for an XML parser

```
int xml_get_current_column_number ( resource parser) \linebreak
```

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_line_number (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

get current line number for an XML parser

```
int xml_get_current_line_number ( resource parser) \linebreak
```

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_error_code (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

get XML parser error code

```
int xml_get_error_code ( resource parser) \linebreak
```

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_parse (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

start parsing an XML document

```
bool xml_parse ( resource parser, string data [, bool is_final]) \linebreak
```

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 *is_final* parameter is set and `TRUE` when the last data is parsed.

is_final (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_parse_into_struct (PHP 3>= 3.0.8, PHP 4 >= 4.0.0)

Parse XML data into an array structure

```
int xml_parse_into_struct ( resource parser, string data, array &values, array &index) \linebreak
```

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 embedded inside a para tag, and then we parse this and 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. parsemolddb.php - parses molddb.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);
}

$db = readDatabase("molddb.xml");
echo "*** Database of AminoAcid objects:\n";
print_r($db);

```

?>

After executing `parsemolddb.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_create (PHP 3>= 3.0.6, PHP 4 >= 4.0.0)

create an XML parser

resource **xml_parser_create** ([string encoding]) \linebreak

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_parser_create_ns (PHP 4 >= 4.0.5)

Create an XML parser

```
resource xml_parser_create_ns ( [string encoding [, string sep]]) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

xml_parser_free (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Free an XML parser

```
bool xml_parser_free ( resource parser) \linebreak
```

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_get_option (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

get options from an XML parser

```
mixed xml_parser_get_option ( resource parser, int option) \linebreak
```

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.

xml_parser_set_option (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set options in an XML parser

```
bool xml_parser_set_option ( resource parser, int option, mixed value) \linebreak
```

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
<code>XML_OPTION_CASE_FOLDING</code>	Integer	Controls whether case-folding is enabled for this XML parser. Enabled by default.
<code>XML_OPTION_TARGET_ENCODING</code>	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 <code>xml_parser_create()</code> . Supported target encodings are <code>ISO-8859-1</code> , <code>US-ASCII</code> and <code>UTF-8</code> .

xml_set_character_data_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set up character data handler

```
bool xml_set_character_data_handler ( resource parser, string handler) \linebreak
```

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** (resource parser, string data) \linebreak

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_default_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set up default handler

bool **xml_set_default_handler** (resource parser, string handler) \linebreak

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** (resource parser, string data) \linebreak

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_element_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set up start and end element handlers

```
bool xml_set_element_handler ( resource parser, string start_element_handler, string end_element_handler)
\linebreak
```

Sets the element handler functions for the XML parser *parser*. *start_element_handler* and *end_element_handler* are strings containing the names of functions that must exist when `xml_parse()` is called for *parser*.

The function named by *start_element_handler* must accept three parameters:

```
start_element_handler ( resource parser, string name, array attribs) \linebreak
```

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 *end_element_handler* must accept two parameters:

```
end_element_handler ( resource parser, string name) \linebreak
```

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_end_namespace_decl_handler (PHP 4 >= 4.0.5)

Set up character data handler

```
bool xml_set_end_namespace_decl_handler ( resource pind, string hdl) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_external_entity_ref_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set up external entity reference handler

```
bool xml_set_external_entity_ref_handler ( resource parser, string handler) \linebreak
```

Sets the external entity reference 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`. ***handler*** (resource parser, string open_entity_names, string base, string system_id, string public_id) \linebreak

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

open_entity_names

The second parameter, *open_entity_names*, 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 (*system_id*) of the external entity. Currently this parameter will always be set to an empty string.

system_id

The fourth parameter, *system_id*, is the system identifier as specified in the entity declaration.

public_id

The fifth parameter, *public_id*, 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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_notation_decl_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

set up notation declaration handler

`bool xml_set_notation_decl_handler (resource parser, string handler) \linebreak`

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 {system_id |
        public_id}>
```

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** (resource parser, string notation_name, string base, string system_id, string public_id) \linebreak

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

notation_name

This is the notation's *name*, as per the notation format described above.

base

This is the base for resolving the system identifier (*system_id*) of the notation declaration. Currently this parameter will always be set to an empty string.

system_id

System identifier of the external notation declaration.

public_id

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

xml_set_object (PHP 4 >= 4.0.0)

Use XML Parser within an object

```
void xml_set_object ( resource parser, object &object) \linebreak
```

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_processing_instruction_handler (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Set up processing instruction (PI) handler

bool **xml_set_processing_instruction_handler** (resource parser, string handler) \linebreak

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** (resource parser, string target, string data) \linebreak

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

`xml_set_start_namespace_decl_handler` (PHP 4 >= 4.0.5)

Set up character data handler

```
bool xml_set_start_namespace_decl_handler ( resource pind, string hdl) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

`xml_set_unparsed_entity_decl_handler` (PHP 3 >= 3.0.6, PHP 4 >= 4.0.0)

Set up unparsed entity declaration handler

```
bool xml_set_unparsed_entity_decl_handler ( resource parser, string handler) \linebreak
```

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** (resource parser, string entity_name, string base, string system_id, string public_id, string notation_name) \linebreak

parser

The first parameter, *parser*, is a reference to the XML parser calling the handler.

entity_name

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.

system_id

System identifier for the external entity.

public_id

Public identifier for the external entity.

notation_name

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.

Note: Instead of a function name, an array containing an object reference and a method name can also be supplied.

CIV. XMLRPC functions

Warning

This extension is *EXPERIMENTAL*. The behaviour of this extension, including the names of its functions, and anything else documented about this extension may change in a future release of PHP without notice. Be warned and use this extension at your own risk.

xmlrpc_decode (PHP 4 >= 4.1.0)

Decodes XML into native PHP types

array **xmlrpc_decode** (string xml [, string encoding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_decode_request (PHP 4 >= 4.1.0)

Decodes XML into native PHP types

array **xmlrpc_decode_request** (string xml, string method [, string encoding]) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_encode (PHP 4 >= 4.1.0)

Generates XML for a PHP value

string **xmlrpc_encode** (mixed value) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_encode_request (PHP 4 >= 4.1.0)

Generates XML for a method request

string **xmlrpc_encode_request** (string method, mixed params) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_get_type (PHP 4 >= 4.1.0)

Gets xmlrpc type for a PHP value. Especially useful for base64 and datetime strings

string **xmlrpc_get_type** (mixed value) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_parse_method_descriptions (PHP 4 >= 4.1.0)

Decodes XML into a list of method descriptions

array **xmlrpc_parse_method_descriptions** (string xml) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_add_introspection_data (PHP 4 >= 4.1.0)

Adds introspection documentation

int **xmlrpc_server_add_introspection_data** (resource server, array desc) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_call_method (PHP 4 >= 4.1.0)

Parses XML requests and call methods

mixed **xmlrpc_server_call_method** (resource server, string xml, mixed user_data [, array output_options])
 \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_create (PHP 4 >= 4.1.0)

Creates an xmlrpc server

resource **xmlrpc_server_create** (void) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_destroy (PHP 4 >= 4.1.0)

Destroys server resources

```
void xmlrpc_server_destroy ( resource server) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_register_introspection_callback (PHP 4 >= 4.1.0)

Register a PHP function to generate documentation

```
bool xmlrpc_server_register_introspection_callback ( resource server, string function) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_server_register_method (PHP 4 >= 4.1.0)

Register a PHP function to resource method matching method_name

```
bool xmlrpc_server_register_method ( resource server, string method_name, string function) \linebreak
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

xmlrpc_set_type (PHP 4 >= 4.1.0)

Sets xmlrpc type, base64 or datetime, for a PHP string value

bool **xmlrpc_set_type** (string value, string type) \linebreak

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, the name of this function, and anything else documented about this function may change in a future release of PHP without notice. Be warned and use this function at your own risk.

Warning

This function is currently not documented; only the argument list is available.

CV. 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 Sablotron 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 `--enable-xslt --with-xslt-sablot` options. The Sablotron library should be installed somewhere your compiler can find it.

About This Extension

This PHP extension provides a processor independent API to XSLT transformations. Currently this extension only supports the Sablotron library from the Ginger Alliance. Support is planned for other libraries, such as the Xalan library or the libxslt library.

Note: This extension is different than the sablotron extension distributed with versions of PHP prior to PHP 4.1, currently only the new XSLT extension in PHP 4.1 is supported. If you need support for the old extension, please ask your questions on the php-general@lists.php.net mailing list.

xslt_create (PHP 4 >= 4.0.3)

Create a new XSLT processor.

resource **xslt_create** (void) \linebreak

Create and return a new XSLT processor resource for manipulation by the other XSLT functions.

xslt_errno (PHP 4 >= 4.0.3)

Return a error number

int **xslt_errno** (resource xh) \linebreak

Return an error code describing the last error that occurred on the passed XSLT processor.

xslt_error (PHP 4 >= 4.0.3)

Return a error string

mixed **xslt_error** (resource xh) \linebreak

Return a string describing the last error that occurred on the passed XSLT processor.

Example 1. Handling errors using the xslt_error() and xslt_errno() functions.

```
<?php

$xh = xslt_create();
$result = xslt_process($xh, 'dog.xml', 'pets.xsl');
if (!$result) {
    die(sprintf("Cannot process XSLT document [%d]: %s",
                xslt_errno($xh), xslt_error($xh)));
}

print($result);

xslt_free($xh);
?>
```

xslt_free (PHP 4 >= 4.0.3)

Free XSLT processor

```
void xslt_free ( resource xh ) \linebreak
```

Free the XSLT processor identified by the given handle.

xslt_process (PHP 4 >= 4.0.3)

Perform an XSLT transformation

```
mixed xslt_process ( resource xh, string xml, string xsl [, string result [, array arguments [, array parameters]])  
\linebreak
```

The `xslt_process()` function is the crux of the new XSLT extension. It allows you to perform an XSLT transformation using almost any type of input source. This is accomplished through the use of argument buffers -- a concept taken from the Sablotron XSLT processor (currently the only XSLT processor this extension supports).

The simplest type of transformation with the `xslt_process()` function is the transformation of an XML file with an XSLT file, placing the result in a third file containing the new XML (or HTML) document. Doing this with sablotron is really quite easy...

Example 1. Using the `xslt_process()` to transform an XML file and a XSL file to a new XML file

```
<?php

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document
if (xslt_process($xh, 'sample.xml', 'sample.xsl', 'result.xml')) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into result.xml";
    print ", result.xml has the following contents\n<br>\n";
    print "<pre>\n";
    readfile('result.xml');
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl into";
    print " result.xml the reason is that " . xslt_error($xh) . " and the ";
    print "error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

While this functionality is great, many times, especially in a web environment, you want to be able to print out your results directly. Therefore, if you omit the third argument to the `xslt_process()` function (or provide a NULL value for the argument), it will automatically return the value of the XSLT transformation, instead of writing it to a file...

Example 2. Using the `xslt_process()` to transform an XML file and a XSL file to a variable containing the resulting XML data

```
<?php

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document, returning the result into the $result variable
$result = xslt_process($xh, 'sample.xml', 'sample.xsl');
if ($result) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into the \$result";
    print " variable, the \$result variable has the following contents\n<br>\n";
    print "<pre>\n";
    print $result;
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl into";
    print " the \$result variable the reason is that " . xslt_error($xh) .
    print " and the error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

The above two cases are the two simplest cases there are when it comes to XSLT transformation and I'd dare say that they are the most common cases, however, sometimes you get your XML and XSLT code from external sources, such as a database or a socket. In these cases you'll have the XML and/or XSLT data in a variable -- and in production applications the overhead of dumping these to file may be too much. This is where XSLT's "argument" syntax, comes to the rescue. Instead of files as the XML and XSLT arguments to the `xslt_process()` function, you can specify "argument place holders" which are then substituted by values given in the arguments array (5th parameter to the `xslt_process()` function). The following is an example of processing XML and XSLT into a result variable without the use of files at all.

Example 3. Using the `xslt_process()` to transform a variable containing XML data and a variable containing XSL data into a variable containing the resulting XML data

```

<?php
// $xml and $xsl contain the XML and XSL data

$arguments = array(
    '/_xml' => $xml,
    '/_xsl' => $xsl
);

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document
$result = xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $arguments);
if ($result) {
    print "SUCCESS, sample.xml was transformed by sample.xsl into the \"$result\";
    print " variable, the \"$result\" variable has the following contents\n<br>\n";
    print "<pre>\n";
    print $result;
    print "</pre>\n";
}
else {
    print "Sorry, sample.xml could not be transformed by sample.xsl into";
    print " the \"$result\" variable the reason is that " . xslt_error($xh) .
    print " and the error code is " . xslt_errno($xh);
}
xslt_free($xh);
?>

```

Finally, the last argument to the `xslt_process()` function is any parameters that you want to pass to the XSLT document. These parameters can then be accessed within your XSL files using the `<xsl:param name="parameter_name">` instruction.

xslt_set_base (PHP 4 >= 4.0.5)

Set the base URI for all XSLT transformations

```
void xslt_set_base ( resource xh, string uri) \linebreak
```

Sets the base URI for all XSLT transformations, the base URI is used with Xpath instructions to resolve document() and other commands which access external resources.

xslt_set_encoding (PHP 4 >= 4.0.5)

Set the encoding for the parsing of XML documents

```
void xslt_set_encoding ( resource xh, string encoding) \linebreak
```

Set the output encoding for the XSLT transformations. When using the Sablotron backend this option is only available when you compile Sablotron with encoding support.

xslt_set_error_handler (PHP 4 >= 4.0.4)

Set an error handler for a XSLT processor

```
void xslt_set_error_handler ( resource xh, mixed handler) \linebreak
```

Set an error handler function for the XSLT processor given by *xh*, this function will be called whenever an error occurs in the XSLT transformation (this function is also called for notices).

xslt_set_log (PHP 4 >= 4.0.6)

Set the log file to write log messages to

```
void xslt_set_log ( resource xh, mixed log) \linebreak
```

xh

A reference to the XSLT parser.

log

This parameter is either a boolean value which toggles logging on and off, or a string containing the logfile in which log errors too.

This function allows you to set the file in which you want XSLT log messages to, XSLT log messages are different than error messages, in that log messages are not actually error messages but rather messages related to the state of the XSLT processor. They are useful for debugging XSLT, when something goes wrong.

By default logging is disabled, in order to enable logging you must first call **xslt_set_log()** with a boolean parameter which enables logging, then if you want to set the log file to debug to, you must then pass it a string containing the filename:

Example 1. Using the XSLT Logging features

```

<?php

$xh = xslt_create();
xslt_set_log($xh, true);
xslt_set_log($xh, getcwd() . 'myfile.log');

$result = xslt_process($xh, 'dog.xml', 'pets.xsl');
print($result);

xslt_free($xh);
?>

```

xslt_set_sax_handler (4.0.3 - 4.0.6 only)

Set SAX handlers for a XSLT processor

```
void xslt_set_sax_handler ( resource xh, array handlers) \linebreak
```

Set SAX handlers on the resource handle given by *xh*. SAX handlers should be a two dimensional array with the format (all top level elements are optional):

```

array(
  [document] =>
    array(
      start document handler,
      end document handler
    ),
  [element] =>
    array(
      start element handler,
      end element handler
    ),
  [namespace] =>
    array(
      start namespace handler,
      end namespace handler
    ),
  [comment] => comment handler,
  [pi] => processing instruction handler,
  [character] => character data handler
)

```

xslt_set_sax_handlers (PHP 4 >= 4.0.6)

Set the SAX handlers to be called when the XML document gets processed

```
void xslt_set_sax_handlers ( resource processor, array handlers) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

xslt_set_scheme_handler (4.0.5 - 4.0.6 only)

Set Scheme handlers for a XSLT processor

```
void xslt_set_scheme_handler ( resource xh, array handlers) \linebreak
```

Set Scheme handlers on the resource handle given by *xh*. Scheme handlers should be an array with the format (all elements are optional):

```
array(
  [get_all] => get all handler,
  [open] => open handler,
  [get] => get handler,
  [put] => put handler,
  [close] => close handler
)
```

xslt_set_scheme_handlers (PHP 4 >= 4.0.6)

Set the scheme handlers for the XSLT processor

```
void xslt_set_scheme_handlers ( resource processor, array handlers) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

CVI. 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 or scans 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.

Example 1. Parallel searching using YAZ()

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 in array host.

```

$num_hosts = count ($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)

Returns additional error information

```
int yaz_addinfo ( int id) \linebreak
```

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_ccl_conf (PHP 4 >= 4.0.5)

Configure CCL parser

```
int yaz_ccl_conf ( int id, array config) \linebreak
```

This function configures the CCL query parser for a target with definitions of access points (CCL qualifiers) and their mapping to RPN. To map a specific CCL query to RPN afterwards call the `yaz_ccl_parse()` function. Each index of the array `config` is the name of a CCL field and the corresponding value holds a string that specifies a mapping to RPN. The mapping is a sequence of attribute-type, attribute-value pairs. Attribute-type and attribute-value is separated by an equal sign (=). Each pair is separated by white space.

Example 1. CCL configuration

In the example below, the CCL parser is configured to support three CCL fields: `ti`, `au` and `isbn`. Each field is mapped to their BIB-1 equivalent. It is assumed that variable `$id` is a target ID.

```
$field["ti"] = "1=4";
$field["au"] = "1=1";
$field["isbn"] = "1=7";
yaz_ccl_conf($id,$field);
```

yaz_ccl_parse (PHP 4 >= 4.0.5)

Invoke CCL Parser

```
int yaz_ccl_parse ( int id, string query, array & result) \linebreak
```

This function invokes the CCL parser. It converts a given CCL FIND query to an RPN query which may be passed to the `yaz_search()` function to perform a search. To define a set of valid CCL fields call `yaz_ccl_conf()` prior to this function. If the supplied `query` was successfully converted to RPN, this

function returns `TRUE`, and the index `rpn` of the supplied array `result` holds a valid RPN query. If the query could not be converted (because of invalid syntax, unknown field, etc.) this function returns `FALSE` and three indexes are set in the resulting array to indicate the cause of failure: `errorCodeCCL` error code (integer), `errorstringCCL` error string, and `errorpos` approximate position in query of failure (integer is character position).

yaz_close (PHP 4)

Closes a YAZ connection

```
int yaz_close ( int id) \linebreak
```

Closes the Z-association given by *id*. The *id* is a target ID as returned by a previous `yaz_connect()` command.

yaz_connect (PHP 4)

Prepares for a connection and Z-association to a Z39.50 target.

```
int yaz_connect ( string zurl [, mixed options]) \linebreak
```

This function returns a positive ID on success; zero on failure.

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.

If the second argument, *options*, is given as a string it is treated as the Z39.50 V2 authentication string (OpenAuth).

If *options* is given as an array the contents of the array serves as options. Note that array options are only supported for PHP 4.1.0 and later.

yaz_connect() options

user

Username for authentication.

group

Group for authentication.

password

Password for authentication.

cookie

Cookie for session (YAZ proxy).

proxy

Proxy for connection (YAZ proxy).

persistent

A boolean. If `TRUE` the connection is persistent; If `FALSE` the connection is not persistent. By default connections are persistent.

piggyback

A boolean. If `TRUE` piggyback is enabled for searches; If `FALSE` piggyback is disabled. By default piggyback is enabled. Enabling piggyback is more efficient and usually saves a network-round-trip for first time fetches of records. However, a few Z39.50 targets doesn't support piggyback or they ignore element set names. For those, piggyback should be disabled.

yaz_database (PHP 4 >= 4.0.6)

Specifies the databases within a session

`int yaz_database (int id, string databases) \linebreak`

This function specifies one or more databases to be used in search, retrieval, etc. - overriding databases specified in call to `yaz_connect()`. Multiple databases are separated by a plus sign `+`.

This function allows you to use different sets of databases within a session.

Returns `TRUE` on success; `FALSE` on error.

yaz_element (PHP 4)

Specifies Element-Set Name for retrieval

`int yaz_element (int id, string elementset) \linebreak`

This function is used in conjunction with `yaz_search()` and `yaz_present()` 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_errno (PHP 4)

Returns error number

```
int yaz_errno ( int id) \linebreak
```

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 (such as when the target closed connection, etc).

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)

Returns error description

```
string yaz_error ( int id) \linebreak
```

Returns error message for target (last request). An empty string is returned if last operation was a success.

yaz_error() returns an english text message corresponding to the last error number as returned by **yaz_errno()**.

yaz_hits (PHP 4)

Returns number of hits for last search

```
int yaz_hits ( int id) \linebreak
```

yaz_hits() returns number of hits for last search.

yaz_itemorder (PHP 4 >= 4.0.5)

Prepares for Z39.50 Item Order with an ILL-Request package

```
int yaz_itemorder ( array args) \linebreak
```

This function prepares for an Extended Services request using the Profile for the Use of Z39.50 Item Order Extended Service to Transport ILL (Profile/1). See this (<http://www.nlc-bnc.ca/iso/ill/stanprf.htm>) and the specification (<http://www.nlc-bnc.ca/iso/ill/document/standard/z-ill-1a.pdf>). The args parameter must be a hash array with information about the Item Order request to be sent. The key of the hash is the name of the corresponding ASN.1 tag path. For example, the ISBN below the Item-ID has the key item-id,ISBN.

The ILL-Request parameters are:

protocol-version-num
 transaction-id,initial-requester-id,person-or-institution-symbol,person
 transaction-id,initial-requester-id,person-or-institution-symbol,institution
 transaction-id,initial-requester-id,name-of-person-or-institution,name-of-person
 transaction-id,initial-requester-id,name-of-person-or-institution,name-of-institution
 transaction-id,transaction-group-qualifier
 transaction-id,transaction-qualifier
 transaction-id,sub-transaction-qualifier
 service-date-time,this,date
 service-date-time,this,time
 service-date-time,original,date
 service-date-time,original,time
 requester-id,person-or-institution-symbol,person
 requester-id,person-or-institution-symbol,institution
 requester-id,name-of-person-or-institution,name-of-person
 requester-id,name-of-person-or-institution,name-of-institution
 responder-id,person-or-institution-symbol,person
 responder-id,person-or-institution-symbol,institution
 responder-id,name-of-person-or-institution,name-of-person
 responder-id,name-of-person-or-institution,name-of-institution
 transaction-type
 delivery-address,postal-address,name-of-person-or-institution,name-of-person
 delivery-address,postal-address,name-of-person-or-institution,name-of-institution
 delivery-address,postal-address,extended-postal-delivery-address
 delivery-address,postal-address,street-and-number
 delivery-address,postal-address,post-office-box
 delivery-address,postal-address,city
 delivery-address,postal-address,region
 delivery-address,postal-address,country
 delivery-address,postal-address,postal-code
 delivery-address,electronic-address,telecom-service-identifier
 delivery-address,electronic-address,telecom-service-address
 billing-address,postal-address,name-of-person-or-institution,name-of-person
 billing-address,postal-address,name-of-person-or-institution,name-of-institution
 billing-address,postal-address,extended-postal-delivery-address
 billing-address,postal-address,street-and-number
 billing-address,postal-address,post-office-box
 billing-address,postal-address,city
 billing-address,postal-address,region
 billing-address,postal-address,country
 billing-address,postal-address,postal-code
 billing-address,electronic-address,telecom-service-identifier
 billing-address,electronic-address,telecom-service-address
 ill-service-type
 requester-optional-messages,can-send-RECEIVED
 requester-optional-messages,can-send-RETURNED
 requester-optional-messages,requester-SHIPPED
 requester-optional-messages,requester-CHECKED-IN

search-type,level-of-service
search-type,need-before-date
search-type,expiry-date
search-type,expiry-flag
place-on-hold
client-id,client-name
client-id,client-status
client-id,client-identifier
item-id,item-type
item-id,call-number
item-id,author
item-id,title
item-id,sub-title
item-id,sponsoring-body
item-id,place-of-publication
item-id,publisher
item-id,series-title-number
item-id,volume-issue
item-id,edition
item-id,publication-date
item-id,publication-date-of-component
item-id,author-of-article
item-id,title-of-article
item-id,pagination
item-id,ISBN
item-id,ISSN
item-id,additional-no-letters
item-id,verification-reference-source
copyright-complicance
retry-flag
forward-flag
requester-note
forward-note

There are also a few parameters that are part of the Extended Services Request package and the ItemOrder package:

package-name
user-id
contact-name
contact-phone
contact-email
itemorder-item

yaz_present (PHP 4 >= 4.0.5)

Prepares for retrieval (Z39.50 present).

```
int yaz_present ( void) \linebreak
```

This function prepares for retrieval of records after a successful search. The `yaz_range()` should be called prior to this function to specify the range of records to be retrieved.

yaz_range (PHP 4)

Specifies the maximum number of records to retrieve

```
int yaz_range ( int id, int start, int number) \linebreak
```

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)

Returns a record

```
int yaz_record ( int id, int pos, string type) \linebreak
```

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_scan (PHP 4 >= 4.0.5)

Prepares for a scan

```
int yaz_scan ( int id, string type, string startterm [, array flags]) \linebreak
```

This function prepares for a Z39.50 Scan Request. Argument `id` specifies target ID. Starting term point for the scan is given by `startterm`. The form in which is the starting term is specified is given by `type`. Currently `type rpn` is supported. The optional `flags` specifies additional information to control the behaviour of the scan request. Three indexes are currently read from the flags: `number` (number of

terms requested), `position` (preferred position of term) and `stepSize` (preferred step size). To actually transfer the Scan Request to the target and receive the Scan Response, `yaz_wait()` must be called. Upon completion of `yaz_wait()` call `yaz_error()` and `yaz_scan_result()` to handle the response.

The syntax of `startterm` is similar to the RPN query as described in `yaz_search()`. The `startterm` consists of zero or more `@attr-operator` specifications, then followed by exactly one token.

Example 1. PHP function that scans titles

```
function scan_titles($id, $startterm) {
    yaz_scan($id, "rpn", "@attr 1=4 " . $startterm);
    yaz_wait();
    $errno = yaz_errno($id);
    if ($errno == 0) {
        $ar = yaz_scan_result($id, &$options);
        echo 'Scan ok; ';
        $ar = yaz_scan_result($id, &$options);
        while(list($key, $val)=each($options)) {
            echo "$key = $val &nbsp;";
        }
        echo '<br><table><tr><td>';
        while(list($key, list($k, $term, $tcount))=each($ar)) {
            if (empty($k)) continue;
            echo "<tr><td>$term</td><td>";
            echo $tcount;
            echo "</td></tr>";
        }
        echo '</table>';
    } else {
        echo "Scan failed. Error: " . yaz_error($id) . "<br>";
    }
}
```

yaz_scan_result (PHP 4 >= 4.0.5)

Returns Scan Response result

array **yaz_scan_result** (int id [, array & result]) \linebreak

Given a target ID this function returns an array with terms as received from the target in the last Scan Response. This function returns an array (0..n-1) where n is the number of terms returned. Each value is a pair where first item is term, second item is result-count. If the *result* is given it will be modified to hold additional information taken from the Scan Response: *number* (number of entries returned), *stepsize* (Step-size), *position* (position of term), *status* (Scan Status).

yaz_search (PHP 4)

Prepares for a search

```
int yaz_search ( int id, string type, string query) \linebreak
```

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 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.

Example 1. Query Examples

Query

```
computer
```

matches documents where "computer" occur. No attributes are specified.

The Query

```
"donald knuth"
```

matches documents where "donald knuth" occur.

For the query

```
@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.

Another more complex one:

```
@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_sort (PHP 4 >= 4.1.0)

Sets sorting criteria

```
int yaz_sort ( int id, string criteria) \linebreak
```

This function sets sorting criteria and enables Z39.50 Sort. Use this function together with `yaz_search()` or `yaz_present()`. Using this function alone doesn't have any effect. If used in conjunction with `yaz_search()` a Z39.50 Sort will be sent after a search response has been received and before any records are retrieved with Z39.50 Present. The *criteria* takes the form

```
field1 flags1 field2 flags2 ...
```

where `field1` specifies primary attributes for sort, `field2` seconds, etc.. The field specifies either numerical attribute combinations consisting of `type=value` pairs separated by comma (e.g. `1=4, 2=1`); or the field may specify a plain string criteria (e.g. `title`). The flags is a sequence of the following characters which may not be separated by any white space.

Sort Flags

a

Sort ascending

d

Sort descending

i

Case insensitive sorting

s

Case sensitive sorting

Example 1. Sort Criterias

To sort on Bib1 attribute title, case insensitive, and ascending you'd use the following sort criteria:

```
1=4 ia
```

If the secondary sorting criteria should be author, case sensitive and ascending you'd use:

```
1=4 ia 1=1003 sa
```


yaz_syntax (PHP 4)

Specifies the preferred record syntax for retrieval.

```
int yaz_syntax ( int id, string syntax) \linebreak
```

The syntax is specified as an OID (Object Identifier) in a raw dot-notation (like 1.2.840.10003.5.10) or as one of the known registered record syntaxes (sutrs, usmarc, grs1, xml, etc.). This function is used in conjunction with `yaz_search()` and `yaz_present()` to specify the preferred record syntax for retrieval.

yaz_wait (PHP 4)

Wait for Z39.50 requests to complete

```
int yaz_wait ( [ array options]) \linebreak
```

This function carries out networked (blocked) activity for outstanding requests which have been prepared by the functions `yaz_connect()`, `yaz_search()`, `yaz_present()`, `yaz_scan()` and `yaz_itemorder()`. **yaz_wait()** returns when all targets have either completed all requests or aborted (in case of errors).

If the *options* array is given that holds options that change the behaviour of **yaz_wait()**.

`timeout`

Sets timeout in seconds. If a target hasn't responded within the timeout it is considered dead and **yaz_wait()** returns. The default value for timeout is 15 seconds.

CVII. 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_all (PHP 4 >= 4.0.6)

Traverse the map and call a function on each entry

```
void yp_all ( string domain, string map, string callback) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

yp_cat (PHP 4 >= 4.0.6)

Return an array containing the entire map

```
array yp_cat ( string domain, string map) \linebreak
```

Warning

This function is currently not documented; only the argument list is available.

yp_err_string (PHP 4 >= 4.0.6)

Returns the error string associated with the previous operation

```
string yp_err_string ( void) \linebreak
```

yp_err_string() returns the error message associated with the previous operation. Useful to indicate what exactly went wrong.

Example 1. Example for NIS errors

```
<?php
    echo "Error: " . yp_err_string();
?>
```

See also `yp_errno()`.

yp_errno (PHP 4 >= 4.0.6)

Returns the error code of the previous operation

`int yp_errno (void) \linebreak`

`yp_errno()` returns the error code of the previous operation.

Possible errors are:

- 1 args to function are bad
- 2 RPC failure - domain has been unbound
- 3 can't bind to server on this domain
- 4 no such map in server's domain
- 5 no such key in map
- 6 internal yp server or client error
- 7 resource allocation failure
- 8 no more records in map database
- 9 can't communicate with portmapper
- 10 can't communicate with ypbind
- 11 can't communicate with ypserv
- 12 local domain name not set
- 13 yp database is bad
- 14 yp version mismatch
- 15 access violation
- 16 database busy

See also `yp_err_string()`.

yp_first (PHP 3 >= 3.0.7, PHP 4 >= 4.0.0)

Returns the first key-value pair from the named map

`array yp_first (string domain, string map) \linebreak`

`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 = $entry ["key"];
$value = $entry ["value"];
echo "First entry in this map has key " . $key . " and value " . $value;
?>
```

See also `yp-get-default-domain()`

yp_get_default_domain (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Fetches the machine's default NIS domain

int **yp_get_default_domain** (void) \linebreak

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_master (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Returns the machine name of the master NIS server for a map

string **yp_master** (string domain, string map) \linebreak

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 >= 4.0.0)

Returns the matched line

string **yp_match** (string domain, string map, string key) \linebreak

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_next (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Returns the next key-value pair in the named map.

array **yp_next** (string domain, string map, string key) \linebreak

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 "No more entries found\n";
    <!-- 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()`.

yp_order (PHP 3>= 3.0.7, PHP 4 >= 4.0.0)

Returns the order number for a map

```
int yp_order ( string domain, string map) \linebreak
```

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: " . $number;
?>

```

See also `yp-get-default-domain()`.

CVIII. Zip File Functions (Read Only Access)

This module uses the functions of the ZZIPlib (<http://zziplib.sourceforge.net/>) library by Guido Draheim to transparently read ZIP compressed archives and the files inside them.

Please note that ZZIPlib only provides a subset of functions provided in a full implementation of the ZIP compression algorithm and can only read ZIP file archives. A normal ZIP utility is needed to create the ZIP file archives read by this library.

Zip support in PHP is not enabled by default. You will need to use the `--with-zip` configuration option when compiling PHP to enable zip support. This module requires ZZIPlib version `>= 0.10.6`.

Note: Zip support before PHP 4.1.0 is experimental. This section reflects the Zip extension as it exists in PHP 4.1.0 and later.

Example Usage

This example opens a ZIP file archive, reads each file in the archive and prints out its contents. The `test2.zip` archive used in this example is one of the test archives in the ZZIPlib source distribution.

Example 1. Zip Usage Example

```
<?php

$zip = zip_open("/tmp/test2.zip");

if ($zip) {

    while ($zip_entry = zip_read($zip)) {
        echo "Name:           " . zip_entry_name($zip_entry) . "\n";
        echo "Actual Filesize:  " . zip_entry_filesize($zip_entry) . "\n";
        echo "Compressed Size:      " . zip_entry_compressedsize($zip_entry) . "\n";
        echo "Compression Method:  " . zip_entry_compressionmethod($zip_entry) . "\n";

        if (zip_entry_open($zip, $zip_entry, "r")) {
            echo "File Contents:\n";
            $buf = zip_entry_read($zip_entry, zip_entry_filesize($zip_entry));
            echo "$buf\n";

            zip_entry_close($zip_entry);
        }
        echo "\n";
    }

    zip_close($zip);
}
```


?>

zip_close (PHP 4 >= 4.1.0)

Close a Zip File Archive

```
void zip_close ( resource zip) \linebreak
```

Closes a zip file archive. The parameter *zip* must be a zip archive previously opened by `zip_open()`.

This function has no return value.

See also `zip_open()` and `zip_read()`.

zip_entry_close (PHP 4 >= 4.1.0)

Close a Directory Entry

```
void zip_entry_close ( resource zip_entry) \linebreak
```

Closes a directory entry specified by *zip_entry*. The parameter *zip_entry* must be a valid directory entry opened by `zip_entry_open()`.

This function has no return value.

See also `zip_entry_open()` and `zip_entry_read()`.

zip_entry_compressedsize (PHP 4 >= 4.1.0)

Retrieve the Compressed Size of a Directory Entry

```
int zip_entry_compressedsize ( resource zip_entry) \linebreak
```

Returns the compressed size of the directory entry specified by *zip_entry*. The parameter *zip_entry* is a valid directory entry returned by `zip_read()`.

See also `zip_open()` and `zip_read()`.

zip_entry_compressionmethod (PHP 4 >= 4.1.0)

Retrieve the Compression Method of a Directory Entry

```
string zip_entry_compressionmethod ( resource zip_entry) \linebreak
```

Returns the compression method of the directory entry specified by *zip_entry*. The parameter *zip_entry* is a valid directory entry returned by `zip_read()`.

See also `zip_open()` and `zip_read()`.

zip_entry_filesize (PHP 4 >= 4.1.0)

Retrieve the Actual File Size of a Directory Entry

int **zip_entry_filesize** (resource zip_entry) \linebreak

Returns the actual size of the directory entry specified by *zip_entry*. The parameter *zip_entry* is a valid directory entry returned by *zip_read()*.

See also *zip_open()* and *zip_read()*.

zip_entry_name (PHP 4 >= 4.1.0)

Retrieve the Name of a Directory Entry

string **zip_entry_name** (resource zip_entry) \linebreak

Returns the name of the directory entry specified by *zip_entry*. The parameter *zip_entry* is a valid directory entry returned by *zip_read()*.

See also *zip_open()* and *zip_read()*.

zip_entry_open (PHP 4 >= 4.1.0)

Open a Directory Entry for Reading

bool **zip_entry_open** (resource zip, resource zip_entry [, string mode]) \linebreak

Opens a directory entry in a zip file for reading. The parameter *zip* is a valid resource handle returned by *zip_open()*. The parameter *zip_entry* is a directory entry resource returned by *zip_read()*. The optional parameter *mode* can be any of the modes specified in the documentaion for *fopen()*.

Note: Currently, *mode* is ignored and is always "rb". This is due to the fact that zip support in PHP is read only access. Please see *fopen()* for an explanation of various modes, including "rb".

Returns TRUE on success, FALSE on failure.

Note: Unlike *fopen()* and other similar functions, the return value of **zip_entry_open()** only indicates the result of the operation and is not needed for reading or closing the directory entry.

See also *zip_entry_read()* and *zip_entry_close()*.

zip_entry_read (PHP 4 >= 4.1.0)

Read From an Open Directory Entry

string **zip_entry_read** (resource zip_entry [, int length]) \linebreak

Reads up to *length* bytes from an open directory entry. If *length* is not specified, then **zip_entry_read()** will attempt to read 1024 bytes. The parameter *zip_entry* is a valid directory entry returned by **zip_read()**.

Note: The *length* parameter should be the uncompressed length you wish to read.

Returns the data read, or `FALSE` if the end of the file is reached.

See also **zip_entry_open()**, **zip_entry_close()** and **zip_entry_filesize()**.

zip_open (PHP 4 >= 4.1.0)

Open a Zip File Archive

resource **zip_open** (string filename) \linebreak

Opens a new zip archive for reading. The *filename* parameter is the filename of the zip archive to open.

Returns a resource handle for later use with **zip_read()** and **zip_close()** or returns `FALSE` if *filename* does not exist.

See also **zip_read()** and **zip_close()**.

zip_read (PHP 4 >= 4.1.0)

Read Next Entry in a Zip File Archive

resource **zip_read** (resource zip) \linebreak

Reads the next entry in a zip file archive. The parameter *zip* must be a zip archive previously opened by **zip_open()**.

Returns a directory entry resource for later use with the **zip_entry_...()** functions.

See also **zip_open()**, **zip_close()**, **zip_entry_open()**, and **zip_entry_read()**.

CIX. Zlib Compression Functions

This module uses the functions of zlib (<http://www.gzip.org/zlib/>) by Jean-loup Gailly and Mark Adler to transparently read and write gzip (.gz) compressed files. You have to use a zlib version $\geq 1.0.9$ 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: Version 4.0.4 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().

In version 4.3.0, this special prefix has been changed to 'zlib:/' to prevent ambiguities with filenames containing ':'.

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 >= 4.0.0)

Close an open gz-file pointer

```
int gzclose ( int zp) \linebreak
```

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()`.

gzcompress (PHP 4)

Compress a string

```
string gzcompress ( string data [, int level]) \linebreak
```

This function returns a compressed version of the input *data* using the ZLIB data format, 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.

For details on the ZLIB compression algorithm see the document "ZLIB Compressed Data Format Specification version 3.3 (<http://www.ietf.org/rfc/rfc1950.txt>)" (RFC 1950).

Note: This is *not* the same as gzip compression, which includes some header data. See `gzencode()` for gzip compression.

See also `gzdeflate()`, `gzinflate()`, `gzuncompress()`, `gzencode()`.

gzdeflate (PHP 4 >= 4.0.4)

Deflate a string

```
string gzdeflate ( string data [, int level]) \linebreak
```

This function returns a compressed version of the input *data* using the DEFLATE data format, 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.

For details on the DEFLATE compression algorithm see the document "DEFLATE Compressed Data Format Specification version 1.3 (<http://www.ietf.org/rfc/rfc1951.txt>)" (RFC 1951).

See also `gzinflate()`, `gzcompress()`, `gzuncompress()`, `gzencode()`.

gzencode (PHP 4 >= 4.0.4)

Create a gzip compressed string

```
string gzencode ( string data [, int level [, int encoding_mode]]) \linebreak
```

This function returns a compressed version of the input *data* compatible with the output of the **gzip** program, 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, if not given the default compression level will be the default compression level of the zlib library.

You can also give **FORCE_GZIP** (the default) or **FORCE_DEFLATE** as optional third parameter *encoding_mode*. If you use **FORCE_DEFLATE**, you get a standard zlib deflated string (inclusive zlib headers) after the gzip file header but without the trailing crc32 checksum.

Note: *level* was added in PHP 4.2, before PHP 4.2 **gzencode()** only had the *data* and (optional) *encoding_mode* parameters..

The resulting data contains the appropriate headers and data structure to make a standard .gz file, e.g.:

Example 1. Creating a gzip file

```
<?php
    $data = implode("", file("bigfile.txt"));
    $gzdata = gzencode($data, 9);
    $fp = fopen("bigfile.txt.gz", "w");
    fwrite($fp, $gzdata);
    fclose($fp);
?>
```

For more information on the GZIP file format, see the document: GZIP file format specification version 4.3 (<http://www.ietf.org/rfc/rfc1952.txt>) (RFC 1952).

See also `gzcompress()`, `gzuncompress()`, `gzdeflate()`, `gzinflate()`.

gzeof (PHP 3, PHP 4 >= 4.0.0)

Test for end-of-file on a gz-file pointer

```
int gzeof ( int zp) \linebreak
```

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 >= 4.0.0)

Read entire gz-file into an array

array **gzfile** (string filename [, int use_include_path]) \linebreak

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 >= 4.0.0)

Get character from gz-file pointer

string **gzgetc** (int zp) \linebreak

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 >= 4.0.0)

Get line from file pointer

string **gzgets** (int zp, int length) \linebreak

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 >= 4.0.0)

Get line from gz-file pointer and strip HTML tags

```
string gzgetss ( int zp, int length [, string allowable_tags]) \linebreak
```

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()`.

gzinflate (PHP 4 >= 4.0.4)

Inflate a deflated string

```
string gzinflate ( string data [, int length]) \linebreak
```

This function takes `data` compressed by `gzdeflate()` 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()`, `gzuncompress()`, `gzdeflate()`, `gzencode()`.

gzopen (PHP 3, PHP 4 >= 4.0.0)

Open gz-file

```
int gzopen ( string filename, string mode [, int use_include_path]) \linebreak
```

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 >= 4.0.0)

Output all remaining data on a gz-file pointer

```
int gzpassthru ( int zp ) \linebreak
```

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 >= 4.0.0)

Write to a gz-file pointer

```
int gzputs ( int zp, string str [, int length] ) \linebreak
```

`gzputs()` is an alias to `gzwrite()`, and is identical in every way.

gzread (PHP 3, PHP 4 >= 4.0.0)

Binary-safe gz-file read

```
string gzread ( int zp, int length ) \linebreak
```

`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";
$d = gzopen ($filename, "r");
$content = gzread ($d, 10000);
gzclose ($d);
```

See also `gzwrite()`, `gzopen()`, `gzgets()`, `gzgetss()`, `gzfile()`, and `gzpassthru()`.

gzrewind (PHP 3, PHP 4 >= 4.0.0)

Rewind the position of a gz-file pointer

`int gzrewind (int zp) \linebreak`

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 >= 4.0.0)

Seek on a gz-file pointer

`int gzseek (int zp, int offset) \linebreak`

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 `gztell()` and `gzrewind()`.

gztell (PHP 3, PHP 4 >= 4.0.0)

Tell gz-file pointer read/write position

`int gztell (int zp) \linebreak`

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()`.

gzuncompress (PHP 4)

Uncompress a deflated string

string **gzuncompress** (string *data* [, int *length*]) \linebreak

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 `gzdeflate()`, `gzinflate()`, `gzcompress()`, `gzencode()`.

gzwrite (PHP 3, PHP 4 >= 4.0.0)

Binary-safe gz-file write

int **gzwrite** (int *zp*, string *string* [, int *length*]) \linebreak

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 >= 4.0.0)

Output a gz-file

int **readgzfile** (string *filename* [, int *use_include_path*]) \linebreak

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()`.

Part V. Extending PHP 4.0

Chapter 24. Overview

"Extending PHP" is easier said than done. PHP has evolved to a full-fledged tool consisting of a few megabytes of source code, and to hack a system like this quite a few things have to be learned and considered. When structuring this chapter, we finally decided on the "learn by doing" approach. This is not the most scientific and professional approach, but the method that's the most fun and gives the best end results. In the following sections, you'll learn quickly how to get the most basic extensions to work almost instantly. After that, you'll learn about Zend's advanced API functionality. The alternative would have been to try to impart the functionality, design, tips, tricks, etc. as a whole, all at once, thus giving a complete look at the big picture before doing anything practical. Although this is the "better" method, as no dirty hacks have to be made, it can be very frustrating as well as energy- and time-consuming, which is why we've decided on the direct approach.

Note that even though this chapter tries to impart as much knowledge as possible about the inner workings of PHP, it's impossible to really give a complete guide to extending PHP that works 100% of the time in all cases. PHP is such a huge and complex package that its inner workings can only be understood if you make yourself familiar with it by practicing, so we encourage you to work with the source.

What Is Zend? and What Is PHP?

The name *Zend* refers to the language engine, PHP's core. The term *PHP* refers to the complete system as it appears from the outside. This might sound a bit confusing at first, but it's not that complicated (see Figure 24-1). To implement a Web script interpreter, you need three parts:

1. The *interpreter* part analyzes the input code, translates it, and executes it.
2. The *functionality* part implements the functionality of the language (its functions, etc.).
3. The *interface* part talks to the Web server, etc.

Zend takes part 1 completely and a bit of part 2; PHP takes parts 2 and 3. Together they form the complete PHP package. Zend itself really forms only the language core, implementing PHP at its very basics with some predefined functions. PHP contains all the modules that actually create the language's outstanding capabilities.

Figure 24-1. The internal structure of PHP.

The following sections discuss where PHP can be extended and how it's done.

Chapter 25. Extension Possibilities

As shown in Figure 24-1 above, PHP can be extended primarily at three points: external modules, built-in modules, and the Zend engine. The following sections discuss these options.

External Modules

External modules can be loaded at script runtime using the function `dl()`. This function loads a shared object from disk and makes its functionality available to the script to which it's being bound. After the script is terminated, the external module is discarded from memory. This method has both advantages and disadvantages, as described in the following table:

Advantages	Disadvantages
External modules don't require recompiling of PHP.	The shared objects need to be loaded every time a script is being executed (every hit), which is very slow.
The size of PHP remains small by "outsourcing" certain functionality.	External additional files clutter up the disk.
	Every script that wants to use an external module's functionality has to specifically include a call to <code>dl()</code> , or the <code>extension</code> tag in <code>php.ini</code> needs to be modified (which is not always a suitable solution).

To sum up, external modules are great for third-party products, small additions to PHP that are rarely used, or just for testing purposes. To develop additional functionality quickly, external modules provide the best results. For frequent usage, larger implementations, and complex code, the disadvantages outweigh the advantages.

Third parties might consider using the `extension` tag in `php.ini` to create additional external modules to PHP. These external modules are completely detached from the main package, which is a very handy feature in commercial environments. Commercial distributors can simply ship disks or archives containing only their additional modules, without the need to create fixed and solid PHP binaries that don't allow other modules to be bound to them.

Built-in Modules

Built-in modules are compiled directly into PHP and carried around with every PHP process; their functionality is instantly available to every script that's being run. Like external modules, built-in modules have advantages and disadvantages, as described in the following table:

Advantages	Disadvantages
No need to load the module specifically; the functionality is instantly available.	Changes to built-in modules require recompiling of PHP.
No external files clutter up the disk; everything resides in the PHP binary.	The PHP binary grows and consumes more memory.

Built-in modules are best when you have a solid library of functions that remains relatively unchanged, requires better than poor-to-average performance, or is used frequently by many scripts on your site. The need to recompile PHP is quickly compensated by the benefit in speed and ease of use. However, built-in modules are not ideal when rapid development of small additions is required.

The Zend Engine

Of course, extensions can also be implemented directly in the Zend engine. This strategy is good if you need a change in the language behavior or require special functions to be built directly into the language core. In general, however, modifications to the Zend engine should be avoided. Changes here result in incompatibilities with the rest of the world, and hardly anyone will ever adapt to specially patched Zend engines. Modifications can't be detached from the main PHP sources and are overridden with the next update using the "official" source repositories. Therefore, this method is generally considered bad practice and, due to its rarity, is not covered in this book.

Chapter 26. Source Layout

Note: Prior to working through the rest of this chapter, you should retrieve clean, unmodified source trees of your favorite Web server. We're working with Apache (available at <http://www.apache.org/>) and, of course, with PHP (available at <http://www.php.net/> - does it need to be said?).

Make sure that you can compile a working PHP environment by yourself! We won't go into this issue here, however, as you should already have this most basic ability when studying this chapter.

Before we start discussing code issues, you should familiarize yourself with the source tree to be able to quickly navigate through PHP's files. This is a must-have ability to implement and debug extensions.

After extracting the PHP archive, you'll see a directory layout similar to that in Figure 26-1.

Figure 26-1. Main directory layout of the PHP source tree.

The following table describes the contents of the major directories.

Directory	Contents
php-4	Main PHP source files and main header files; here you'll find all of PHP's API definitions, macros, etc. (important)
ext	Repository for dynamic and built-in modules; by default, these are the "official" PHP modules that have been installed
pear	Directory for the PHP class repository. At the time of this writing, this is still in the design phase, but it's being tested
sapi	Contains the code for the different server abstraction layers.
TSRM	Location of the "Thread Safe Resource Manager" (TSRM) for Zend and PHP.
zend	Location of Zend's file; here you'll find all of Zend's API definitions, macros, etc. (important).

Discussing all the files included in the PHP package is beyond the scope of this chapter. However, you should take a close look at the following files:

- `php.h`, located in the main PHP directory. This file contains most of PHP's macro and API definitions.
- `zend.h`, located in the main Zend directory. This file contains most of Zend's macros and definitions.
- `zend_API.h`, also located in the Zend directory, which defines Zend's API.

You should also follow some sub-inclusions from these files; for example, the ones relating to the Zend executor, the PHP initialization file support, and such. After reading these files, take the time to navigate around the package a little to see the interdependencies of all files and modules - how they relate to each

other and especially how they make use of each other. This also helps you to adapt to the coding style in which PHP is authored. To extend PHP, you should quickly adapt to this style.

Extension Conventions

Zend is built using certain conventions; to avoid breaking its standards, you should follow the rules described in the following sections.

Macros

For almost every important task, Zend ships predefined macros that are extremely handy. The tables and figures in the following sections describe most of the basic functions, structures, and macros. The macro definitions can be found mainly in `zend.h` and `zend_API.h`. We suggest that you take a close look at these files after having studied this chapter. (Although you can go ahead and read them now, not everything will make sense to you yet.)

Memory Management

Resource management is a crucial issue, especially in server software. One of the most valuable resources is memory, and memory management should be handled with extreme care. Memory management has been partially abstracted in Zend, and you should stick to this abstraction for obvious reasons: Due to the abstraction, Zend gets full control over all memory allocations. Zend is able to determine whether a block is in use, automatically freeing unused blocks and blocks with lost references, and thus prevent memory leaks. The functions to be used are described in the following table:

Function	Description
emalloc()	Serves as replacement for malloc() .
efree()	Serves as replacement for free() .
estrdup()	Serves as replacement for strdup() .
estrndup()	Serves as replacement for strndup() . Faster than estrdup() and binary-safe. This is the recommended function.
ecalloc()	Serves as replacement for calloc() .
erealloc()	Serves as replacement for realloc() .

emalloc(), **estrdup()**, **estrndup()**, **ecalloc()**, and **erealloc()** allocate internal memory; **efree()** frees these previously allocated blocks. Memory handled by the **e*()** functions is considered local to the current process and is discarded as soon as the script executed by this process is terminated.

Warning

To allocate resident memory that survives termination of the current script, you can use **malloc()** and **free()**. This should only be done with extreme care, however, and only in conjunction with demands of the Zend API; otherwise, you risk memory leaks.

Zend also features a thread-safe resource manager to provide better native support for multithreaded Web servers. This requires you to allocate local structures for all of your global variables to allow concurrent threads to be run. Because the thread-safe mode of Zend was not finished back when this was written, it is not yet extensively covered here.

Directory and File Functions

The following directory and file functions should be used in Zend modules. They behave exactly like their C counterparts, but provide virtual working directory support on the thread level.

Zend Function	Regular C Function
V_GETCWD()	getcwd()
V_FOPEN()	fopen()
V_OPEN()	open()
V_CHDIR()	chdir()
V_GETWWD()	getwd()
V_CHDIR_FILE()	Takes a file path as an argument and changes the current working directory to that file's directory.
V_STAT()	stat()
V_LSTAT()	lstat()

String Handling

Strings are handled a bit differently by the Zend engine than other values such as integers, Booleans, etc., which don't require additional memory allocation for storing their values. If you want to return a string from a function, introduce a new string variable to the symbol table, or do something similar, you have to make sure that the memory the string will be occupying has previously been allocated, using the aforementioned **e*()** functions for allocation. (This might not make much sense to you yet; just keep it somewhere in your head for now - we'll get back to it shortly.)

Complex Types

Complex types such as arrays and objects require different treatment. Zend features a single API for these types - they're stored using hash tables.

Note: To reduce complexity in the following source examples, we're only working with simple types such as integers at first. A discussion about creating more advanced types follows later in this chapter.

Chapter 27. PHP's Automatic Build System

PHP 4 features an automatic build system that's very flexible. All modules reside in a subdirectory of the `ext` directory. In addition to its own sources, each module consists of an M4 file (for example, see http://www.gnu.org/manual/m4/html_mono/m4.html) for configuration and a `Makefile.in` file, which is responsible for compilation (the results of `autoconf` and `automake`; for example, see <http://sourceware.cygnus.com/autoconf/autoconf.html> and <http://sourceware.cygnus.com/automake/automake.html>).

Both files are generated automatically, along with `.cvsignore`, by a little shell script named `ext_skel` that resides in the `ext` directory. As argument it takes the name of the module that you want to create. The shell script then creates a directory of the same name, along with the appropriate `config.m4` and `Makefile.in` files.

Step by step, the process looks like this:

```
root@dev:/usr/local/src/php4/ext > ./ext_skel my_module
Creating directory
Creating basic files: config.m4 Makefile.in .cvsignore [done].
To use your new extension, you will have to execute the following steps:
  $ cd ..
  $ ./buildconf
  $ ./configure # (your extension is automatically enabled)
  $ vi ext/my_module/my_module.c
  $ make
```

Repeat the last two steps as often as necessary.

This instruction creates the aforementioned files. To include the new module in the automatic configuration and build process, you have to run `buildconf`, which regenerates the `configure` script by searching through the `ext` directory and including all found `config.m4` files.

Finally, running `configure` parses all configuration options and generates a makefile based on those options and the options you specify in `Makefile.in`.

Example 27-1 shows the previously generated `Makefile.in`:

Example 27-1. The default `makefile.in`.

```
# $Id: Extending_Zend_Build.xml,v 1.6 2002/03/25 08:13:46 hholzgra Exp $
LTLIBRARY_NAME      = libmy_module.la
LTLIBRARY_SOURCES   = my_module.c
LTLIBRARY_SHARED_NAME = my_module.la include
$(top_srcdir)/build/dynlib.mk
```

There's not much to tell about this one: It contains the names of the input and output files. You could also specify build instructions for other files if your module is built from multiple source files.

The default `config.m4` shown in Example 27-2 is a bit more complex:

Example 27-2. The default `config.m4`.

```
dnl $Id: Extending_Zend_Build.xml,v 1.6 2002/03/25 08:13:46 hholzgra Exp $
dnl config.m4 for extension my_module
dnl don't forget to call PHP_EXTENSION(my_module)
```

```

dnl If your extension references something external, use with:
PHP_ARG_WITH(my_module, for my_module support,
dnl Make sure that the comment is aligned:
    [ --with-my_module          Include my_module support])
dnl Otherwise use enable:
PHP_ARG_ENABLE(my_module, whether to enable my_module support,
dnl Make sure that the comment is aligned:
    [ --enable-my_module       Enable my_module support])
if test "$PHP_MY_MODULE" != "no"; then
dnl Action..
PHP_EXTENSION(my_module, $ext_shared)
fi

```

If you're unfamiliar with M4 files (now is certainly a good time to get familiar), this might be a bit confusing at first; but it's actually quite easy.

Note: Everything prefixed with `dnl` is treated as a comment and is not parsed.

The `config.m4` file is responsible for parsing the command-line options passed to `configure` at configuration time. This means that it has to check for required external files and do similar configuration and setup tasks.

The default file creates two configuration directives in the `configure` script: `--with-my_module` and `--enable-my_module`. Use the first option when referring external files (such as the `--with-apache` directive that refers to the Apache directory). Use the second option when the user simply has to decide whether to enable your extension. Regardless of which option you use, you should uncomment the other, unnecessary one; that is, if you're using `--enable-my_module`, you should remove support for `--with-my_module`, and vice versa.

By default, the `config.m4` file created by `ext_skel` accepts both directives and automatically enables your extension. Enabling the extension is done by using the `PHP_EXTENSION` macro. To change the default behavior to include your module into the PHP binary when desired by the user (by explicitly specifying `--enable-my_module` or `--with-my_module`), change the test for `$PHP_MY_MODULE` to `== "yes"`:

```

if test "$PHP_MY_MODULE" == "yes"; then dnl
    Action.. PHP_EXTENSION(my_module, $ext_shared)
fi

```

This would require you to use `--enable-my_module` each time when reconfiguring and recompiling PHP.

Note: Be sure to run `buildconf` every time you change `config.m4`!

We'll go into more details on the M4 macros available to your configuration scripts later in this chapter. For now, we'll simply use the default files. The sample sources on the CD-ROM all have working `config.m4` files. To include them into the PHP build process, simply copy the source directories to your PHP `ext` directory, run `buildconf`, and then include the sample modules you want by using the appropriate `--enable-*` directives with `configure`.

Chapter 28. Creating Extensions

We'll start with the creation of a very simple extension at first, which basically does nothing more than implement a function that returns the integer it receives as parameter. Example 28-1 shows the source.

Example 28-1. A simple extension.

```

/* include standard header */
#include "php.h"

/* declaration of functions to be exported */
ZEND_FUNCTION(first_module);

/* compiled function list so Zend knows what's in this module */
zend_function_entry firstmod_functions[] =
{
    ZEND_FE(first_module, NULL)
    {NULL, NULL, NULL}
};

/* compiled module information */
zend_module_entry firstmod_module_entry =
{
    STANDARD_MODULE_HEADER,
    "First Module",
    firstmod_functions,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NO_VERSION_YET,
    STANDARD_MODULE_PROPERTIES
};

/* implement standard "stub" routine to introduce ourselves to Zend */
#ifdef COMPILE_DL_FIRST_MODULE
ZEND_GET_MODULE(firstmod)
#endif

/* implement function that is meant to be made available to PHP */
ZEND_FUNCTION(first_module)
{
    long parameter;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &parameter) == FAILURE) {
        return;
    }

    RETURN_LONG(parameter);
}

```

This code contains a complete PHP module. We'll explain the source code in detail shortly, but first we'd like to discuss the build process. (This will allow the impatient to experiment before we dive into API discussions.)

Note: The example source makes use of some features introduced with the Zend version used in PHP 4.1.0 and above, it won't compile with older PHP 4.0.x versions.

Compiling Modules

There are basically two ways to compile modules:

- Use the provided "make" mechanism in the `ext` directory, which also allows building of dynamic loadable modules.
- Compile the sources manually.

The first method should definitely be favored, since, as of PHP 4.0, this has been standardized into a sophisticated build process. The fact that it is so sophisticated is also its drawback, unfortunately - it's hard to understand at first. We'll provide a more detailed introduction to this later in the chapter, but first let's work with the default files.

The second method is good for those who (for some reason) don't have the full PHP source tree available, don't have access to all files, or just like to juggle with their keyboard. These cases should be extremely rare, but for the sake of completeness we'll also describe this method.

Compiling Using Make. To compile the sample sources using the standard mechanism, copy all their subdirectories to the `ext` directory of your PHP source tree. Then run `buildconf`, which will create an updated `configure` script containing appropriate options for the new extension. By default, all the sample sources are disabled, so you don't have to fear breaking your build process.

After you run `buildconf`, `configure --help` shows the following additional modules:

```
--enable-array_experiments  BOOK: Enables array experiments
--enable-call_userland      BOOK: Enables userland module
--enable-cross_conversion    BOOK: Enables cross-conversion module
--enable-first_module        BOOK: Enables first module
--enable-infoprint           BOOK: Enables infoprint module
--enable-reference_test      BOOK: Enables reference test module
--enable-resource_test       BOOK: Enables resource test module
--enable-variable_creation   BOOK: Enables variable-creation module
```

The module shown earlier in Example 28-1 can be enabled with `--enable-first_module` or `--enable-first_module=yes`.

Compiling Manually. To compile your modules manually, you need the following commands:

Action	Command
Compiling	<code>cc -fPIC -D_COMPILE_DL=1 -I/usr/local/include -I. -I./Zend -c -o <your_object_file> <your_c_file></code>

Linking	<code>cc -shared -L/usr/local/lib -rdynamic -o <your_module_file> <your_object_file(s)></code>
---------	--

The command to compile the module simply instructs the compiler to generate position-independent code (`-fpic` shouldn't be omitted) and additionally defines the constant `COMPILE_DL` to tell the module code that it's compiled as a dynamically loadable module (the test module above checks for this; we'll discuss it shortly). After these options, it specifies a number of standard include paths that should be used as the minimal set to compile the source files.

Note: All include paths in the example are relative to the directory `ext`. If you're compiling from another directory, change the pathnames accordingly. Required items are the PHP directory, the Zend directory, and (if necessary), the directory in which your module resides.

The link command is also a plain vanilla command instructing linkage as a dynamic module.

You can include optimization options in the compilation command, although these have been omitted in this example (but some are included in the makefile template described in an earlier section).

Note: Compiling and linking manually as a static module into the PHP binary involves very long instructions and thus is not discussed here. (It's not very efficient to type all those commands.)

Chapter 29. Using Extensions

Depending on the build process you selected, you should either end up with a new PHP binary to be linked into your Web server (or run as CGI), or with an .so (shared object) file. If you compiled the example file `first_module.c` as a shared object, your result file should be `first_module.so`. To use it, you first have to copy it to a place from which it's accessible to PHP. For a simple test procedure, you can copy it to your `htdocs` directory and try it with the source in Example 29-1. If you compiled it into the PHP binary, omit the call to `dl()`, as the module's functionality is instantly available to your scripts.

Warning

For security reasons, you *should not* put your dynamic modules into publicly accessible directories. Even though it *can* be done and it simplifies testing, you should put them into a separate directory in production environments.

Example 29-1. A test file for `first_module.so`.

```
<?php
// remove next comment if necessary
// dl("first_module.so");

$param = 2;
$return = first_module($param);

print("We sent '$param' and got '$return'");

?>
```

Calling this PHP file in your Web browser should give you the output shown in Figure 29-1.

Figure 29-1. Output of `first_module.php`.

If required, the dynamic loadable module is loaded by calling the `dl()` function. This function looks for the specified shared object, loads it, and makes its functions available to PHP. The module exports the

function **first_module()**, which accepts a single parameter, converts it to an integer, and returns the result of the conversion.

If you've gotten this far, congratulations! You just built your first extension to PHP.

Chapter 30. Troubleshooting

Actually, not much troubleshooting can be done when compiling static or dynamic modules. The only problem that could arise is that the compiler will complain about missing definitions or something similar. In this case, make sure that all header files are available and that you specified their path correctly in the compilation command. To be sure that everything is located correctly, extract a clean PHP source tree and use the automatic build in the `ext` directory with the fresh files; this will guarantee a safe compilation environment. If this fails, try manual compilation.

PHP might also complain about missing functions in your module. (This shouldn't happen with the sample sources if you didn't modify them.) If the names of external functions you're trying to access from your module are misspelled, they'll remain as "unlinked symbols" in the symbol table. During dynamic loading and linkage by PHP, they won't resolve because of the typing errors - there are no corresponding symbols in the main binary. Look for incorrect declarations in your module file or incorrectly written external references. Note that this problem is specific to dynamic loadable modules; it doesn't occur with static modules. Errors in static modules show up at compile time.

Chapter 31. Source Discussion

Now that you've got a safe build environment and you're able to include the modules into PHP files, it's time to discuss how everything works.

Module Structure

All PHP modules follow a common structure:

- Header file inclusions (to include all required macros, API definitions, etc.)
- C declaration of exported functions (required to declare the Zend function block)
- Declaration of the Zend function block
- Declaration of the Zend module block
- Implementation of `get_module()`
- Implementation of all exported functions

Header File Inclusions

The only header file you really have to include for your modules is `php.h`, located in the PHP directory. This file makes all macros and API definitions required to build new modules available to your code.

Tip: It's good practice to create a separate header file for your module that contains module-specific definitions. This header file should contain all the forward definitions for exported functions and include `php.h`. If you created your module using `ext_skel` you already have such a header file prepared.

Declaring Exported Functions

To declare functions that are to be exported (i.e., made available to PHP as new native functions), Zend provides a set of macros. A sample declaration looks like this:

```
ZEND_FUNCTION ( my_function );
```

`ZEND_FUNCTION` declares a new C function that complies with Zend's internal API. This means that the function is of type `void` and accepts `INTERNAL_FUNCTION_PARAMETERS` (another macro) as parameters. Additionally, it prefixes the function name with `zif`. The immediately expanded version of the above definitions would look like this:

```
void zif_my_function ( INTERNAL_FUNCTION_PARAMETERS );
```

Expanding `INTERNAL_FUNCTION_PARAMETERS` results in the following:

```

void zif_my_function( int ht
                    , zval * return_value
                    , zval * this_ptr
                    , int return_value_used
                    , zend_executor_globals * executor_globals
                    );

```

Since the interpreter and executor core have been separated from the main PHP package, a second API defining macros and function sets has evolved: the Zend API. As the Zend API now handles quite a few of the responsibilities that previously belonged to PHP, a lot of PHP functions have been reduced to macros aliasing to calls into the Zend API. The recommended practice is to use the Zend API wherever possible, as the old API is only preserved for compatibility reasons. For example, the types `zval` and `pval` are identical. `zval` is Zend's definition; `pval` is PHP's definition (actually, `pval` is an alias for `zval` now). As the macro `INTERNAL_FUNCTION_PARAMETERS` is a Zend macro, the above declaration contains `zval`. When writing code, you should always use `zval` to conform to the new Zend API.

The parameter list of this declaration is very important; you should keep these parameters in mind (see Table 31-1 for descriptions).

Table 31-1. Zend's Parameters to Functions Called from PHP

Parameter	Description
<code>ht</code>	The number of arguments passed to the Zend function. You should not touch this directly, but instead use <code>zend_get_parameters_count()</code> .
<code>return_value</code>	This variable is used to pass any return values of your function back to PHP. Access to this variable is by <code>zend_update_return_value()</code> .
<code>this_ptr</code>	Using this variable, you can gain access to the object in which your function is contained, if it's used with <code>INTERNAL_FUNCTION_PARAMETERS</code> .
<code>return_value_used</code>	This flag indicates whether an eventual return value from this function will actually be used by the caller.
<code>executor_globals</code>	This variable points to global settings of the Zend engine. You'll find this useful when creating new variables.

Declaration of the Zend Function Block

Now that you have declared the functions to be exported, you also have to introduce them to Zend. Introducing the list of functions is done by using an array of `zend_function_entry`. This array consecutively contains all functions that are to be made available externally, with the function's name as it should appear in PHP and its name as defined in the C source. Internally, `zend_function_entry` is defined as shown in Example 31-1.

Example 31-1. Internal declaration of `zend_function_entry`.

```

typedef struct _zend_function_entry {
    char *fname;
    void (*handler)(INTERNAL_FUNCTION_PARAMETERS);
    unsigned char *func_arg_types;
};

```

```
} zend_function_entry;
```

Entry	Description
fname	Denotes the function name as seen in PHP (for example, <code>fopen</code> , <code>mysql_connect</code> , or, in our example, <code>first_module</code>).
handler	Pointer to the C function responsible for handling calls to this function. For example, see the standard macro <code>ZEND_FE</code> .
func_arg_types	Allows you to mark certain parameters so that they're forced to be passed by reference. You usually should use <code>ZEND_FE</code> for this.

In the example above, the declaration looks like this:

```
zend_function_entry firstmod_functions[] =
{
    ZEND_FE(first_module, NULL)
    {NULL, NULL, NULL}
};
```

You can see that the last entry in the list always has to be `{NULL, NULL, NULL}`. This marker has to be set for Zend to know when the end of the list of exported functions is reached.

Note: You *cannot* use the predefined macros for the end marker, as these would try to refer to a function named "NULL"!

The macro `ZEND_FE` (short for 'Zend Function Entry') simply expands to a structure entry in `zend_function_entry`. Note that these macros introduce a special naming scheme to your functions - your C functions will be prefixed with `zif_`, meaning that `ZEND_FE(first_module)` will refer to a C function `zif_first_module()`. If you want to mix macro usage with hand-coded entries (not a good practice), keep this in mind.

Tip: Compilation errors that refer to functions named `zif_*` relate to functions defined with `ZEND_FE`.

Table 31-2 shows a list of all the macros that you can use to define functions.

Table 31-2. Macros for Defining Functions

Macro Name	Description
<code>ZEND_FE(name, arg_types)</code>	Defines a function entry of the name <code>name</code> in <code>zend_function_entry</code> .
<code>ZEND_NAMED_FE/php_name, name, arg_types)</code>	Defines a function that will be available to PHP by the name <code>php_name</code> .
<code>ZEND_FALIAS(name, alias, arg_types)</code>	Defines an alias named <code>alias</code> for <code>name</code> . <code>arg_types</code> needs to be set.
<code>PHP_FE(name, arg_types)</code>	Old PHP API equivalent of <code>ZEND_FE</code> .
<code>PHP_NAMED_FE(runtime_name, name, arg_types)</code>	Old PHP API equivalent of <code>ZEND_NAMED_FE</code> .

Note: You can't use `ZEND_FE` in conjunction with `PHP_FUNCTION`, or `PHP_FE` in conjunction with `ZEND_FUNCTION`. However, it's perfectly legal to mix `ZEND_FE` and `ZEND_FUNCTION` with `PHP_FE` and `PHP_FUNCTION` when staying with the same macro set for each function to be declared. But mixing is *not* recommended; instead, you're advised to use the `ZEND_*` macros only.

Declaration of the Zend Module Block

This block is stored in the structure `zend_module_entry` and contains all necessary information to describe the contents of this module to Zend. You can see the internal definition of this module in Example 31-2.

Example 31-2. Internal declaration of `zend_module_entry`.

```
typedef struct _zend_module_entry zend_module_entry;

    struct _zend_module_entry {
        unsigned short size;
        unsigned int zend_api;
        unsigned char zend_debug;
        unsigned char zts;
        char *name;
        zend_function_entry *functions;
        int (*module_startup_func)(INIT_FUNC_ARGS);
        int (*module_shutdown_func)(SHUTDOWN_FUNC_ARGS);
        int (*request_startup_func)(INIT_FUNC_ARGS);
        int (*request_shutdown_func)(SHUTDOWN_FUNC_ARGS);
        void (*info_func)(ZEND_MODULE_INFO_FUNC_ARGS);
        char *version;
        int (*global_startup_func)(void);
        int (*global_shutdown_func)(void);

        [ Rest of the structure is not interesting here ]

    };
```

Entry	Description
size, zend_api, zend_debug and zts	Usually filled with the "STANDARD_MODULE_HEADER", which fills these four members.
name	Contains the module name (for example, "File functions", "Socket functions").
functions	Points to the Zend function block, discussed in the preceding section.
module_startup_func	This function is called once upon module initialization and can be used to do one-time initialization.
module_shutdown_func	This function is called once upon module shutdown and can be used to do one-time deinitialization.
request_startup_func	This function is called once upon every page request and can be used to do one-time initialization.
request_shutdown_func	This function is called once after every page request and works as counterpart to request_startup_func.
info_func	When <code>phpinfo()</code> is called in a script, Zend cycles through all loaded modules and calls this function.
version	The version of the module. You can use <code>NO_VERSION_YET</code> if you don't want to give the version.
Remaining structure elements	These are used internally and can be prefilled by using the macro <code>STANDARD_MODULE_HEADER</code> .

In our example, this structure is implemented as follows:

```
zend_module_entry firstmod_module_entry =
{
    STANDARD_MODULE_HEADER,
```

```

    "First Module",
    firstmod_functions,
    NULL, NULL, NULL, NULL, NULL,
    NO_VERSION_YET,
    STANDARD_MODULE_PROPERTIES,
};

```

This is basically the easiest and most minimal set of values you could ever use. The module name is set to `First Module`, then the function list is referenced, after which all startup and shutdown functions are marked as being unused.

For reference purposes, you can find a list of the macros involved in declared startup and shutdown functions in Table 31-3. These are not used in our basic example yet, but will be demonstrated later on. You should make use of these macros to declare your startup and shutdown functions, as these require special arguments to be passed (`INIT_FUNC_ARGS` and `SHUTDOWN_FUNC_ARGS`), which are automatically included into the function declaration when using the predefined macros. If you declare your functions manually and the PHP developers decide that a change in the argument list is necessary, you'll have to change your module sources to remain compatible.

Table 31-3. Macros to Declare Startup and Shutdown Functions

Macro	Description
<code>ZEND_MINIT(module)</code>	Declares a function for module startup. The generated name will be <code>zend_minit_<module></code> .
<code>ZEND_MSHUTDOWN(module)</code>	Declares a function for module shutdown. The generated name will be <code>zend_mshutdown_<module></code> .
<code>ZEND_RINIT(module)</code>	Declares a function for request startup. The generated name will be <code>zend_rinit_<module></code> .
<code>ZEND_RSHUTDOWN(module)</code>	Declares a function for request shutdown. The generated name will be <code>zend_rshutdown_<module></code> .
<code>ZEND_MINFO(module)</code>	Declares a function for printing module information, used when <code>phpinfo()</code> is called. The generated name will be <code>zend_minfo_<module></code> .

Creation of `get_module()`

This function is special to all dynamic loadable modules. Take a look at the creation via the `ZEND_GET_MODULE` macro first:

```

#ifdef COMPILE_DL_FIRSTMOD
    ZEND_GET_MODULE(firstmod)
#endif

```

The function implementation is surrounded by a conditional compilation statement. This is needed since the function `get_module()` is only required if your module is built as a dynamic extension. By specifying a definition of `COMPILE_DL_FIRSTMOD` in the compiler command (see above for a discussion of the compilation instructions required to build a dynamic extension), you can instruct your module whether you intend to build it as a dynamic extension or as a built-in module. If you want a built-in module, the implementation of `get_module()` is simply left out.

get_module() is called by Zend at load time of the module. You can think of it as being invoked by the `dl()` call in your script. Its purpose is to pass the module information block back to Zend in order to inform the engine about the module contents.

If you don't implement a **get_module()** function in your dynamic loadable module, Zend will compliment you with an error message when trying to access it.

Implementation of All Exported Functions

Implementing the exported functions is the final step. The example function in `first_module` looks like this:

```
ZEND_FUNCTION(first_module)
{
    long parameter;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &parameter) == FAILURE) {
        return;
    }

    RETURN_LONG(parameter);
}
```

The function declaration is done using `ZEND_FUNCTION`, which corresponds to `ZEND_FE` in the function entry table (discussed earlier).

After the declaration, code for checking and retrieving the function's arguments, argument conversion, and return value generation follows (more on this later).

Summary

That's it, basically - there's nothing more to implementing PHP modules. Built-in modules are structured similarly to dynamic modules, so, equipped with the information presented in the previous sections, you'll be able to fight the odds when encountering PHP module source files.

Now, in the following sections, read on about how to make use of PHP's internals to build powerful extensions.

Chapter 32. Accepting Arguments

One of the most important issues for language extensions is accepting and dealing with data passed via arguments. Most extensions are built to deal with specific input data (or require parameters to perform their specific actions), and function arguments are the only real way to exchange data between the PHP level and the C level. Of course, there's also the possibility of exchanging data using predefined global values (which is also discussed later), but this should be avoided by all means, as it's extremely bad practice.

PHP doesn't make use of any formal function declarations; this is why call syntax is always completely dynamic and never checked for errors. Checking for correct call syntax is left to the user code. For example, it's possible to call a function using only one argument at one time and four arguments the next time - both invocations are syntactically absolutely correct.

Determining the Number of Arguments

Since PHP doesn't have formal function definitions with support for call syntax checking, and since PHP features variable arguments, sometimes you need to find out with how many arguments your function has been called. You can use the `ZEND_NUM_ARGS` macro in this case. In previous versions of PHP, this macro retrieved the number of arguments with which the function has been called based on the function's hash table entry, `ht`, which is passed in the `INTERNAL_FUNCTION_PARAMETERS` list. As `ht` itself now contains the number of arguments that have been passed to the function, `ZEND_NUM_ARGS` has been stripped down to a dummy macro (see its definition in `zend_API.h`). But it's still good practice to use it, to remain compatible with future changes in the call interface. *Note:* The old PHP equivalent of this macro is `ARG_COUNT`.

The following code checks for the correct number of arguments:

```
if (ZEND_NUM_ARGS() != 2) WRONG_PARAM_COUNT;
```

If the function is not called with two arguments, it exits with an error message. The code snippet above makes use of the tool macro `WRONG_PARAM_COUNT`, which can be used to generate a standard error message (see Figure 32-1).

Figure 32-1. `WRONG_PARAM_COUNT` in action.

This macro prints a default error message and then returns to the caller. Its definition can also be found in `zend_API.h` and looks like this:

```
ZEND_API void wrong_param_count(void);
```

```
#define WRONG_PARAM_COUNT { wrong_param_count(); return; }
```

As you can see, it calls an internal function named **wrong_param_count()** that's responsible for printing the warning. For details on generating customized error messages, see the later section "Printing Information."

Retrieving Arguments

New parameter parsing API: This chapter documents the new Zend parameter parsing API introduced by Andrei Zmievski. It was introduced in the development stage between PHP 4.0.6 and 4.1.0 .

Parsing parameters is a very common operation and it may get a bit tedious. It would also be nice to have standardized error checking and error messages. Since PHP 4.1.0, there is a way to do just that by using the new parameter parsing API. It greatly simplifies the process of receiving parameters, but it has a drawback in that it can't be used for functions that expect variable number of parameters. But since the vast majority of functions do not fall into those categories, this parsing API is recommended as the new standard way.

The prototype for parameter parsing function looks like this:

```
int zend_parse_parameters(int num_args TSRMLS_DC, char *type_spec, ...);
```

The first argument to this function is supposed to be the number of actual parameters passed to your function, so `ZEND_NUM_ARGS()` can be used for that. The second parameter should always be `TSRMLS_CC` macro. The third argument is a string that specifies the number and types of arguments your function is expecting, similar to how `printf` format string specifies the number and format of the output values it should operate on. And finally the rest of the arguments are pointers to variables which should receive the values from the parameters.

zend_parse_parameters() also performs type conversions whenever possible, so that you always receive the data in the format you asked for. Any type of scalar can be converted to another one, but conversions between complex types (arrays, objects, and resources) and scalar types are not allowed.

If the parameters could be obtained successfully and there were no errors during type conversion, the function will return `SUCCESS`, otherwise it will return `FAILURE`. The function will output informative error messages, if the number of received parameters does not match the requested number, or if type conversion could not be performed.

Here are some sample error messages:

```
Warning - ini_get_all() requires at most 1 parameter, 2 given
Warning - wddx_deserialize() expects parameter 1 to be string, array given
```

Of course each error message is accompanied by the filename and line number on which it occurs.

Here is the full list of type specifiers:

- l - long
- d - double
- s - string (with possible null bytes) and its length
- b - boolean
- r - resource, stored in `zval*`
- a - array, stored in `zval*`
- o - object (of any class), stored in `zval*`
- O - object (of class specified by class entry), stored in `zval*`
- z - the actual `zval*`

The following characters also have a meaning in the specifier string:

- | - indicates that the remaining parameters are optional. The storage variables corresponding to these parameters should be initialized to default values by the extension, since they will not be touched by the parsing function if the parameters are not passed.
- / - the parsing function will call **SEPARATE_ZVAL_IF_NOT_REF()** on the parameter it follows, to provide a copy of the parameter, unless it's a reference.
- ! - the parameter it follows can be of specified type or `NULL` (only applies to a, o, O, r, and z). If `NULL` value is passed by the user, the storage pointer will be set to `NULL`.

The best way to illustrate the usage of this function is through examples:

```

/* Gets a long, a string and its length, and a zval. */
long l;
char *s;
int s_len;
zval *param;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                        "lzs", &l, &s, &s_len, &param) == FAILURE) {
    return;
}

/* Gets an object of class specified by my_ce, and an optional double. */
zval *obj;
double d = 0.5;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                        "O|d", &obj, my_ce, &d) == FAILURE) {
    return;
}

/* Gets an object or null, and an array.
   If null is passed for object, obj will be set to NULL. */
zval *obj;

```

```

zval *arr;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "O!a", &obj, &arr) == FAIL-
URE) {
    return;
}

/* Gets a separated array. */
zval *arr;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "a/", &arr) == FAILURE) {
    return;
}

/* Get only the first three parameters (useful for varargs functions). */
zval *z;
zend_bool b;
zval *r;
if (zend_parse_parameters(3, "zbr!", &z, &b, &r) == FAILURE) {
    return;
}

```

Note that in the last example we pass 3 for the number of received parameters, instead of `ZEND_NUM_ARGS()`. What this lets us do is receive the least number of parameters if our function expects a variable number of them. Of course, if you want to operate on the rest of the parameters, you will have to use `zend_get_parameters_array_ex()` to obtain them.

The parsing function has an extended version that allows for an additional flags argument that controls its actions.

```
int zend_parse_parameters_ex(int flags, int num_args TSRMLS_DC, char *type_spec, ...);
```

The only flag you can pass currently is `ZEND_PARSE_PARAMS_QUIET`, which instructs the function to not output any error messages during its operation. This is useful for functions that expect several sets of completely different arguments, but you will have to output your own error messages.

For example, here is how you would get either a set of three longs or a string:

```

long l1, l2, l3;
char *s;
if (zend_parse_parameters_ex(ZEND_PARSE_PARAMS_QUIET,
                            ZEND_NUM_ARGS() TSRMLS_CC,
                            "l1l", &l1, &l2, &l3) == SUCCESS) {
    /* manipulate longs */
} else if (zend_parse_parameters_ex(ZEND_PARSE_PARAMS_QUIET,
                                    ZEND_NUM_ARGS(), "s", &s, &s_len) == SUCCESS) {
    /* manipulate string */
} else {
    php_error(E_WARNING, "%s() takes either three long values or a string as argument",

```



```

        get_active_function_name(TSRMLS_C));
    return;
}

```

With all the abovementioned ways of receiving function parameters you should have a good handle on this process. For even more example, look through the source code for extensions that are shipped with PHP - they illustrate every conceivable situation.

Old way of retrieving arguments (deprecated)

Deprecated parameter parsing API: This API is deprecated and superseded by the new ZEND parameter parsing API.

After having checked the number of arguments, you need to get access to the arguments themselves. This is done with the help of `zend_get_parameters_ex()`:

```

zval **parameter;

if(zend_get_parameters_ex(1, &parameter) != SUCCESS)
    WRONG_PARAM_COUNT;

```

All arguments are stored in a zval container, which needs to be pointed to *twice*. The snippet above tries to retrieve one argument and make it available to us via the parameter pointer.

`zend_get_parameters_ex()` accepts at least two arguments. The first argument is the number of arguments to retrieve (which should match the number of arguments with which the function has been called; this is why it's important to check for correct call syntax). The second argument (and all following arguments) are pointers to pointers to pointers to zvals. (Confusing, isn't it?) All these pointers are required because Zend works internally with `**zval`; to adjust a local `**zval` in our function, `zend_get_parameters_ex()` requires a pointer to it.

The return value of `zend_get_parameters_ex()` can either be `SUCCESS` or `FAILURE`, indicating (unsurprisingly) success or failure of the argument processing. A failure is most likely related to an incorrect number of arguments being specified, in which case you should exit with `WRONG_PARAM_COUNT`.

To retrieve more than one argument, you can use a similar snippet:

```

zval **param1, **param2, **param3, **param4;

if(zend_get_parameters_ex(4, &param1, &param2, &param3, &param4) != SUCCESS)
    WRONG_PARAM_COUNT;

```

`zend_get_parameters_ex()` only checks whether you're trying to retrieve too many parameters. If the function is called with five arguments, but you're only retrieving three of them with `zend_get_parameters_ex()`, you won't get an error but will get the first three parameters instead. Subsequent calls of `zend_get_parameters_ex()` won't retrieve the remaining arguments, but will get the same arguments again.

Dealing with a Variable Number of Arguments/Optional Parameters

If your function is meant to accept a variable number of arguments, the snippets just described are sometimes suboptimal solutions. You have to create a line calling `zend_get_parameters_ex()` for every possible number of arguments, which is often unsatisfying.

For this case, you can use the function `zend_get_parameters_array_ex()`, which accepts the number of parameters to retrieve and an array in which to store them:

```
zval **parameter_array[4];

/* get the number of arguments */
argument_count = ZEND_NUM_ARGS();

/* see if it satisfies our minimal request (2 arguments) */
/* and our maximal acceptance (4 arguments) */
if(argument_count < 2 || argument_count > 5)
    WRONG_PARAM_COUNT;

/* argument count is correct, now retrieve arguments */
if(zend_get_parameters_array_ex(argument_count, parameter_array) != SUCCESS)
    WRONG_PARAM_COUNT;
```

First, the number of arguments is checked to make sure that it's in the accepted range. After that, `zend_get_parameters_array_ex()` is used to fill `parameter_array` with valid pointers to the argument values.

A very clever implementation of this can be found in the code handling PHP's `fsockopen()` located in `ext/standard/fsock.c`, as shown in Example 32-1. Don't worry if you don't know all the functions used in this source yet; we'll get to them shortly.

Example 32-1. PHP's implementation of variable arguments in `fsockopen()`.

```
pval **args[5];
```

```

int *sock=emalloc(sizeof(int));
int *sockp;
int arg_count=ARG_COUNT(ht);
int socketd = -1;
unsigned char udp = 0;
struct timeval timeout = { 60, 0 };
unsigned short portno;
unsigned long conv;
char *key = NULL;
FLS_FETCH();

if (arg_count > 5 || arg_count < 2 || zend_get_parameters_array_ex(arg_count, args)==FAILURE)
    CLOSE_SOCKET(1);
    WRONG_PARAM_COUNT;
}

switch(arg_count) {
    case 5:
        convert_to_double_ex(args[4]);
        conv = (unsigned long) (Z_DVAL_P(args[4]) * 1000000.0);
        timeout.tv_sec = conv / 1000000;
        timeout.tv_usec = conv % 1000000;
        /* fall-through */
    case 4:
        if (!PZVAL_IS_REF(*args[3])) {
            php_error(E_WARNING, "error string argument to fsockopen not passed by reference");
        }
        pval_copy_constructor(*args[3]);
        ZVAL_EMPTY_STRING(*args[3]);
        /* fall-through */
    case 3:
        if (!PZVAL_IS_REF(*args[2])) {
            php_error(E_WARNING, "error argument to fsockopen not passed by reference");
            return;
        }
        ZVAL_LONG(*args[2], 0);
        break;
}

convert_to_string_ex(args[0]);
convert_to_long_ex(args[1]);
portno = (unsigned short) Z_LVAL_P(args[1]);

key = emalloc(Z_STRLEN_P(args[0]) + 10);

```

`fsockopen()` accepts two, three, four, or five parameters. After the obligatory variable declarations, the function checks for the correct range of arguments. Then it uses a fall-through mechanism in a `switch()` statement to deal with all arguments. The `switch()` statement starts with the maximum number of arguments being passed (five). After that, it automatically processes the case of four arguments being passed, then three, by omitting the otherwise obligatory `break` keyword in all stages.

After having processed the last case, it exits the `switch()` statement and does the minimal argument processing needed if the function is invoked with only two arguments.

This multiple-stage type of processing, similar to a stairway, allows convenient processing of a variable number of arguments.

Accessing Arguments

To access arguments, it's necessary for each argument to have a clearly defined type. Again, PHP's extremely dynamic nature introduces some quirks. Because PHP never does any kind of type checking, it's possible for a caller to pass any kind of data to your functions, whether you want it or not. If you expect an integer, for example, the caller might pass an array, and vice versa - PHP simply won't notice.

To work around this, you have to use a set of API functions to force a type conversion on every argument that's being passed (see Table 32-1).

Note: All conversion functions expect a `**zval` as parameter.

Table 32-1. Argument Conversion Functions

Function	Description
<code>convert_to_boolean_ex()</code>	Forces conversion to a Boolean type. Boolean values remain untouched. Longs, doubles, and strings are converted to Boolean.
<code>convert_to_long_ex()</code>	Forces conversion to a long, the default integer type. NULL values, Booleans, resources, and strings are converted to long.
<code>convert_to_double_ex()</code>	Forces conversion to a double, the default floating-point type. NULL values, Booleans, resources, and strings are converted to double.
<code>convert_to_string_ex()</code>	Forces conversion to a string. Strings remain untouched. NULL values are converted to an empty string, and other values are converted to their string representation.
<code>convert_to_array_ex(value)</code>	Forces conversion to an array. Arrays remain untouched. Objects are converted to an array of their properties.
<code>convert_to_object_ex(value)</code>	Forces conversion to an object. Objects remain untouched. NULL values are converted to a NULL object.
<code>convert_to_null_ex(value)</code>	Forces the type to become a NULL value, meaning empty.

Note: You can find a demonstration of the behavior in `cross_conversion.php` on the accompanying CD-ROM. Figure 32-2 shows the output.

Figure 32-2. Cross-conversion behavior of PHP.

Using these functions on your arguments will ensure type safety for all data that's passed to you. If the supplied type doesn't match the required type, PHP forces dummy contents on the resulting value (empty strings, arrays, or objects, 0 for numeric values, FALSE for Booleans) to ensure a defined state.

Following is a quote from the sample module discussed previously, which makes use of the conversion functions:

```
zval **parameter;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter) != SUCCESS))
{
    WRONG_PARAM_COUNT;
}

convert_to_long_ex(parameter);

RETURN_LONG(Z_LVAL_P(parameter));
```

After retrieving the parameter pointer, the parameter value is converted to a long (an integer), which also forms the return value of this function. Understanding access to the contents of the value requires a short discussion of the zval type, whose definition is shown in Example 32-2.

Example 32-2. PHP/Zend zval type definition.

```
typedef pval zval;

typedef struct _zval_struct zval;

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

struct _zval_struct {
    /* Variable information */
    zvalue_value value; /* value */
    unsigned char type; /* active type */
    unsigned char is_ref;
    short refcount;
};
```

Actually, `pval` (defined in `php.h`) is only an alias of `zval` (defined in `zend.h`), which in turn refers to `_zval_struct`. This is a most interesting structure. `_zval_struct` is the "master" structure, containing the value structure, type, and reference information. The substructure `zvalue_value` is a union that contains the variable's contents. Depending on the variable's type, you'll have to access different members of this union. For a description of both structures, see Table 32-2, Table 32-3 and Table 32-4.

Table 32-2. Zend `zval` Structure

Entry	Description
<code>value</code>	Union containing this variable's contents. See Table 32-3 for a description.
<code>type</code>	Contains this variable's type. For a list of available types, see Table 32-4.
<code>is_ref</code>	0 means that this variable is not a reference; 1 means that this variable is a reference to another variable.
<code>refcount</code>	The number of references that exist for this variable. For every new reference to the value stored in this variable, the

Table 32-3. Zend `zvalue_value` Structure

Entry	Description
<code>lval</code>	Use this property if the variable is of the type <code>IS_LONG</code> , <code>IS_BOOLEAN</code> , or <code>IS_RESOURCE</code> .
<code>dval</code>	Use this property if the variable is of the type <code>IS_DOUBLE</code> .
<code>str</code>	This structure can be used to access variables of the type <code>IS_STRING</code> . The member <code>len</code> contains the string length; the
<code>ht</code>	This entry points to the variable's hash table entry if the variable is an array.
<code>obj</code>	Use this property if the variable is of the type <code>IS_OBJECT</code> .

Table 32-4. Zend Variable Type Constants

Constant	Description
<code>IS_NULL</code>	Denotes a NULL (empty) value.
<code>IS_LONG</code>	A long (integer) value.
<code>IS_DOUBLE</code>	A double (floating point) value.
<code>IS_STRING</code>	A string.
<code>IS_ARRAY</code>	Denotes an array.
<code>IS_OBJECT</code>	An object.
<code>IS_BOOL</code>	A Boolean value.
<code>IS_RESOURCE</code>	A resource (for a discussion of resources, see the appropriate section below).
<code>IS_CONSTANT</code>	A constant (defined) value.

To access a long you access `zval.value.lval`, to access a double you use `zval.value.dval`, and so on. Because all values are stored in a union, trying to access data with incorrect union members results in meaningless output.

Accessing arrays and objects is a bit more complicated and is discussed later.

Dealing with Arguments Passed by Reference

If your function accepts arguments passed by reference that you intend to modify, you need to take some precautions.

What we didn't say yet is that under the circumstances presented so far, you don't have write access to any zval containers designating function parameters that have been passed to you. Of course, you can change any zval containers that you created within your function, but you mustn't change any zvals that refer to Zend-internal data!

We've only discussed the so-called `*_ex()` API so far. You may have noticed that the API functions we've used are called `zend_get_parameters_ex()` instead of `zend_get_parameters()`, `convert_to_long_ex()` instead of `convert_to_long()`, etc. The `*_ex()` functions form the so-called new "extended" Zend API. They give a minor speed increase over the old API, but as a tradeoff are only meant for providing read-only access.

Because Zend works internally with references, different variables may reference the same value. Write access to a zval container requires this container to contain an isolated value, meaning a value that's not referenced by any other containers. If a zval container were referenced by other containers and you changed the referenced zval, you would automatically change the contents of the other containers referencing this zval (because they'd simply point to the changed value and thus change their own value as well).

`zend_get_parameters_ex()` doesn't care about this situation, but simply returns a pointer to the desired zval containers, whether they consist of references or not. Its corresponding function in the traditional API, `zend_get_parameters()`, immediately checks for referenced values. If it finds a reference, it creates a new, isolated zval container; copies the referenced data into this newly allocated space; and then returns a pointer to the new, isolated value.

This action is called *zval separation* (or *pval separation*). Because the `*_ex()` API doesn't perform zval separation, it's considerably faster, while at the same time disabling write access.

To change parameters, however, write access is required. Zend deals with this situation in a special way: Whenever a parameter to a function is passed by reference, it performs automatic zval separation. This means that whenever you're calling a function like this in PHP, Zend will automatically ensure that `$parameter` is being passed as an isolated value, rendering it to a write-safe state:

```
my_function(&$parameter);
```

But this *is not* the case with regular parameters! All other parameters that are not passed by reference are in a read-only state.

This requires you to make sure that you're really working with a reference - otherwise you might produce unwanted results. To check for a parameter being passed by reference, you can use the macro `PZVAL_IS_REF`. This macro accepts a `zval*` to check if it is a reference or not. Examples are given in in Example 32-3.

Example 32-3. Testing for referenced parameter passing.

```
zval *parameter;
```

```

if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "z", &parameter) == FAILURE)
    return;

/* check for parameter being passed by reference */
if (!PZVAL_IS_REF(*parameter)) {
    {
        zend_error(E_WARNING, "Parameter wasn't passed by reference");
        RETURN_NULL();
    }
}

/* make changes to the parameter */
ZVAL_LONG(*parameter, 10);

```

Assuring Write Safety for Other Parameters

You might run into a situation in which you need write access to a parameter that's retrieved with `zend_get_parameters_ex()` but not passed by reference. For this case, you can use the macro `SEPARATE_ZVAL`, which does a zval separation on the provided container. The newly generated zval is detached from internal data and has only a local scope, meaning that it can be changed or destroyed without implying global changes in the script context:

```

zval **parameter;

/* retrieve parameter */
zend_get_parameters_ex(1, &parameter);

/* at this stage, <parameter> still is connected */
/* to Zend's internal data buffers */

/* make <parameter> write-safe */
SEPARATE_ZVAL(parameter);

/* now we can safely modify <parameter> */
/* without implying global changes */

```


`SEPARATE_ZVAL` uses **`emalloc()`** to allocate the new zval container, which means that even if you don't deallocate this memory yourself, it will be destroyed automatically upon script termination. However, doing a lot of calls to this macro without freeing the resulting containers will clutter up your RAM.

Note: As you can easily work around the lack of write access in the "traditional" API (with **`zend_get_parameters()`** and so on), this API seems to be obsolete, and is not discussed further in this chapter.

Chapter 33. Creating Variables

When exchanging data from your own extensions with PHP scripts, one of the most important issues is the creation of variables. This section shows you how to deal with the variable types that PHP supports.

Overview

To create new variables that can be seen "from the outside" by the executing script, you need to allocate a new zval container, fill this container with meaningful values, and then introduce it to Zend's internal symbol table. This basic process is common to all variable creations:

```
zval *new_variable;

/* allocate and initialize new container */
MAKE_STD_ZVAL(new_variable);

/* set type and variable contents here, see the following sections */

/* introduce this variable by the name "new_variable_name" into the symbol table */
ZEND_SET_SYMBOL(EG(active_symbol_table), "new_variable_name", new_variable);

/* the variable is now accessible to the script by using $new_variable_name */
```

The macro `MAKE_STD_ZVAL` allocates a new zval container using `ALLOC_ZVAL` and initializes it using `INIT_ZVAL`. As implemented in Zend at the time of this writing, *initializing* means setting the reference count to 1 and clearing the `is_ref` flag, but this process could be extended later - this is why it's a good idea to keep using `MAKE_STD_ZVAL` instead of only using `ALLOC_ZVAL`. If you want to optimize for speed (and you don't have to explicitly initialize the zval container here), you can use `ALLOC_ZVAL`, but this isn't recommended because it doesn't ensure data integrity.

`ZEND_SET_SYMBOL` takes care of introducing the new variable to Zend's symbol table. This macro checks whether the value already exists in the symbol table and converts the new symbol to a reference if so (with automatic deallocation of the old zval container). This is the preferred method if speed is not a crucial issue and you'd like to keep memory usage low.

Note that `ZEND_SET_SYMBOL` makes use of the Zend executor globals via the macro `EG`. By specifying `EG(active_symbol_table)`, you get access to the currently active symbol table, dealing with the active, local scope. The local scope may differ depending on whether the function was invoked from within a function.

If you need to optimize for speed and don't care about optimal memory usage, you can omit the check for an existing variable with the same value and instead force insertion into the symbol table by using `zend_hash_update()`:

```
zval *new_variable;

/* allocate and initialize new container */
MAKE_STD_ZVAL(new_variable);

/* set type and variable contents here, see the following sections */
```

```

/* introduce this variable by the name "new_variable_name" into the symbol ta-
ble */
zend_hash_update(
    EG(active_symbol_table),
    "new_variable_name",
    strlen("new_variable_name") + 1,
    &new_variable,
    sizeof(zval *),
    NULL
);

```

This is actually the standard method used in most modules.

The variables generated with the snippet above will always be of local scope, so they reside in the context in which the function has been called. To create new variables in the global scope, use the same method but refer to another symbol table:

```

zval *new_variable;

// allocate and initialize new container
MAKE_STD_ZVAL(new_variable);

//
// set type and variable contents here
//

// introduce this variable by the name "new_variable_name" into the global sym-
bol table
ZEND_SET_SYMBOL(&EG(symbol_table), "new_variable_name", new_variable);

```

The macro `ZEND_SET_SYMBOL` is now being called with a reference to the main, global symbol table by referring `EG(symbol_table)`.

Note: The `active_symbol_table` variable is a pointer, but `symbol_table` is not. This is why you have to use `EG(active_symbol_table)` and `&EG(symbol_table)` as parameters to `ZEND_SET_SYMBOL` - it requires a pointer.

Similarly, to get a more efficient version, you can hardcode the symbol table update:

```

zval *new_variable;

// allocate and initialize new container
MAKE_STD_ZVAL(new_variable);

//
// set type and variable contents here
//

// introduce this variable by the name "new_variable_name" into the global sym-
bol table
zend_hash_update(
    &EG(symbol_table),

```

```

    "new_variable_name",
    strlen("new_variable_name") + 1,
    &new_variable,
    sizeof(zval *),
    NULL
);

```

Example 33-1 shows a sample source that creates two variables - `local_variable` with a local scope and `global_variable` with a global scope (see Figure 9.7). The full example can be found on the CD-ROM.

Note: You can see that the global variable is actually not accessible from within the function. This is because it's not imported into the local scope using `global $global_variable;` in the PHP source.

Example 33-1. Creating variables with different scopes.

```

ZEND_FUNCTION(variable_creation)
{
    zval *new_var1, *new_var2;

    MAKE_STD_ZVAL(new_var1);
    MAKE_STD_ZVAL(new_var2);

    ZVAL_LONG(new_var1, 10);
    ZVAL_LONG(new_var2, 5);

    ZEND_SET_SYMBOL(EG(active_symbol_table), "local_variable", new_var1);
    ZEND_SET_SYMBOL(&EG(symbol_table), "global_variable", new_var2);

    RETURN_NULL();
}

```

Longs (Integers)

Now let's get to the assignment of data to variables, starting with longs. Longs are PHP's integers and are very simple to store. Looking at the `zval.value` container structure discussed earlier in this chapter, you

can see that the long data type is directly contained in the union, namely in the lval field. The corresponding type value for longs is `IS_LONG` (see Example 33-2).

Example 33-2. Creation of a long.

```
zval *new_long;

MAKE_STD_ZVAL(new_long);

new_long->type = IS_LONG;
new_long->value.lval = 10;
```

Alternatively, you can use the macro `ZVAL_LONG`:

```
zval *new_long;

MAKE_STD_ZVAL(new_long);
ZVAL_LONG(new_long, 10);
```

Doubles (Floats)

Doubles are PHP's floats and are as easy to assign as longs, because their value is also contained directly in the union. The member in the `zval.value` container is `dval`; the corresponding type is `IS_DOUBLE`.

```
zval *new_double;

MAKE_STD_ZVAL(new_double);

new_double->type = IS_DOUBLE;
new_double->value.dval = 3.45;
```

Alternatively, you can use the macro `ZVAL_DOUBLE`:

```
zval *new_double;

MAKE_STD_ZVAL(new_double);
ZVAL_DOUBLE(new_double, 3.45);
```

Strings

Strings need slightly more effort. As mentioned earlier, all strings that will be associated with Zend's internal data structures need to be allocated using Zend's own memory-management functions. Referencing of static strings or strings allocated with standard routines is not allowed. To assign strings, you have to access the structure `str` in the `zval.value` container. The corresponding type is `IS_STRING`:

```
zval *new_string;
char *string_contents = "This is a new string variable";

MAKE_STD_ZVAL(new_string);

new_string->type = IS_STRING;
new_string->value.str.len = strlen(string_contents);
new_string->value.str.val = estrdup(string_contents);
```

Note the usage of Zend's `estrdup()` here. Of course, you can also use the predefined macro `ZVAL_STRING`:

```
zval *new_string;
char *string_contents = "This is a new string variable";

MAKE_STD_ZVAL(new_string);
ZVAL_STRING(new_string, string_contents, 1);
```

`ZVAL_STRING` accepts a third parameter that indicates whether the supplied string contents should be duplicated (using `estrdup()`). Setting this parameter to 1 causes the string to be duplicated; 0 simply uses the supplied pointer for the variable contents. This is most useful if you want to create a new variable referring to a string that's already allocated in Zend internal memory.

If you want to truncate the string at a certain position or you already know its length, you can use `ZVAL_STRINGL(zval, string, length, duplicate)`, which accepts an explicit string length to be set for the new string. This macro is faster than `ZVAL_STRING` and also binary-safe.

To create empty strings, set the string length to 0 and use `empty_string` as contents:

```
new_string->type = IS_STRING;
new_string->value.str.len = 0;
new_string->value.str.val = empty_string;
```

Of course, there's a macro for this as well (`ZVAL_EMPTY_STRING`):

```
MAKE_STD_ZVAL(new_string);
ZVAL_EMPTY_STRING(new_string);
```

Booleans

Booleans are created just like longs, but have the type `IS_BOOL`. Allowed values in `lval` are 0 and 1:

```
zval *new_bool;

MAKE_STD_ZVAL(new_bool);

new_bool->type = IS_BOOL;
new_bool->value.lval = 1;
```

The corresponding macros for this type are `ZVAL_BOOL` (allowing specification of the value) as well as `ZVAL_TRUE` and `ZVAL_FALSE` (which explicitly set the value to `TRUE` and `FALSE`, respectively).

Arrays

Arrays are stored using Zend's internal hash tables, which can be accessed using the `zend_hash_*()` API. For every array that you want to create, you need a new hash table handle, which will be stored in the `ht` member of the `zval.value` container.

There's a whole API solely for the creation of arrays, which is extremely handy. To start a new array, you call `array_init()`.

```
zval *new_array;

MAKE_STD_ZVAL(new_array);

if(array_init(new_array) != SUCCESS)
{
    // do error handling here
}
```

If `array_init()` fails to create a new array, it returns `FAILURE`.

To add new elements to the array, you can use numerous functions, depending on what you want to do. Table 33-1, Table 33-2 and Table 33-3 describe these functions. All functions return `FAILURE` on failure and `SUCCESS` on success.

Table 33-1. Zend's API for Associative Arrays

Function	Description
<code>add_assoc_long(zval *array, char *key, long n);()</code>	Adds an element of type <code>long</code> .
<code>add_assoc_unset(zval *array, char *key);()</code>	Adds an unset element.
<code>add_assoc_bool(zval *array, char *key, int b);()</code>	Adds a Boolean element.
<code>add_assoc_resource(zval *array, char *key, int r);()</code>	Adds a resource to the array.

add_assoc_double (zval *array, char *key, double d);()	Adds a floating-point value.
add_assoc_string (zval *array, char *key, char *str, int duplicate);()	Adds a string to the array. The flag duplicate specifies whether the string contents have to be copied to Zend internal memory.
add_assoc_stringl (zval *array, char *key, char *str, uint length, int duplicate); ()	Adds a string with the desired length length to the array. Otherwise, behaves like add_assoc_string () .

Table 33-2. Zend's API for Indexed Arrays, Part 1

Function	Description
add_index_long (zval *array, uint idx, long n);()	Adds an element of type <code>long</code> .
add_index_unset (zval *array, uint idx);()	Adds an unset element.
add_index_bool (zval *array, uint idx, int b);()	Adds a Boolean element.
add_index_resource (zval *array, uint idx, int r);()	Adds a resource to the array.
add_index_double (zval *array, uint idx, double d);()	Adds a floating-point value.
add_index_string (zval *array, uint idx, char *str, int duplicate);()	Adds a string to the array. The flag duplicate specifies whether the string contents have to be copied to Zend internal memory.
add_index_stringl (zval *array, uint idx, char *str, uint length, int duplicate);()	Adds a string with the desired length length to the array. This function is faster and binary-safe. Otherwise, behaves like add_index_string () .

Table 33-3. Zend's API for Indexed Arrays, Part 2

Function	Description
add_next_index_long (zval *array, long n);()	Adds an element of type <code>long</code> .
add_next_index_unset (zval *array);()	Adds an unset element.
add_next_index_bool (zval *array, int b);()	Adds a Boolean element.
add_next_index_resource (zval *array, int r);()	Adds a resource to the array.
add_next_index_double (zval *array, double d);()	Adds a floating-point value.
add_next_index_string (zval *array, char *str, int duplicate);()	Adds a string to the array. The flag duplicate specifies whether the string contents have to be copied to Zend internal memory.
add_next_index_stringl (zval *array, char *str, uint length, int duplicate);()	Adds a string with the desired length length to the array. This function is faster and binary-safe. Otherwise, behaves like add_index_string () .

All these functions provide a handy abstraction to Zend's internal hash API. Of course, you can also use the hash functions directly - for example, if you already have a zval container allocated that you want to

insert into an array. This is done using `zend_hash_update()` for associative arrays (see Example 33-3) and `zend_hash_index_update()` for indexed arrays (see Example 33-4):

Example 33-3. Adding an element to an associative array.

```
zval *new_array, *new_element;
char *key = "element_key";

MAKE_STD_ZVAL(new_array);
MAKE_STD_ZVAL(new_element);

if(array_init(new_array) == FAILURE)
{
    // do error handling here
}

ZVAL_LONG(new_element, 10);

if(zend_hash_update(new_array->value.ht, key, strlen(key) + 1, (void *)&new_element, sizeof(
```

Example 33-4. Adding an element to an indexed array.

```
zval *new_array, *new_element;
int key = 2;

MAKE_STD_ZVAL(new_array);
MAKE_STD_ZVAL(new_element);

if(array_init(new_array) == FAILURE)
{
    // do error handling here
}

ZVAL_LONG(new_element, 10);

if(zend_hash_index_update(new_array->value.ht, key, (void *)&new_element, sizeof(zval *), NU
```

To emulate the functionality of `add_next_index_*()`, you can use this:

```
zend_hash_next_index_insert(ht, zval **new_element, sizeof(zval *), NULL)
```

Note: To return arrays from a function, use **array_init()** and all following actions on the predefined variable `return_value` (given as argument to your exported function; see the earlier discussion of the call interface). You do not have to use `MAKE_STD_ZVAL` on this.

Tip: To avoid having to write `new_array->value.ht` every time, you can use `HASH_OF(new_array)`, which is also recommended for compatibility and style reasons.

Objects

Since objects can be converted to arrays (and vice versa), you might have already guessed that they have a lot of similarities to arrays in PHP. Objects are maintained with the same hash functions, but there's a different API for creating them.

To initialize an object, you use the function **object_init()**:

```
zval *new_object;

MAKE_STD_ZVAL(new_object);

if(object_init(new_object) != SUCCESS)
{
    // do error handling here
}
```

You can use the functions described in Table 33-4 to add members to your object.

Table 33-4. Zend's API for Object Creation

Function	Description
<code>add_property_long(zval *object, char *key, long l);()</code>	Adds a long to the object.
<code>add_property_unset(zval *object, char *key);()</code>	Adds an unset property to the object.
<code>add_property_bool(zval *object, char *key, int b);()</code>	Adds a Boolean to the object.
<code>add_property_resource(zval *object, char *key, long r);()</code>	Adds a resource to the object.
<code>add_property_double(zval *object, char *key, double d);()</code>	Adds a double to the object.
<code>add_property_string(zval *object, char *key, char *str, int duplicate);()</code>	Adds a string to the object.
<code>add_property_stringl(zval *object, char *key, char *str, uint length, int duplicate);()</code>	Adds a string of the specified length to the object.
<code>add_property_zval(zval *object, char *key, zval *container);()</code>	Adds a <code>zval</code> container to the object.

Resources

Resources are a special kind of data type in PHP. The term *resources* doesn't really refer to any special kind of data, but to an abstraction method for maintaining any kind of information. Resources are kept in a special resource list within Zend. Each entry in the list has a corresponding type definition that denotes the kind of resource to which it refers. Zend then internally manages all references to this

resource. Access to a resource is never possible directly - only via a provided API. As soon as all references to a specific resource are lost, a corresponding shutdown function is called.

For example, resources are used to store database links and file descriptors. The *de facto* standard implementation can be found in the MySQL module, but other modules such as the Oracle module also make use of resources.

Note: In fact, a resource can be a pointer to anything you need to handle in your functions (e.g. pointer to a structure) and the user only has to pass a single resource variable to your function.

To create a new resource you need to register a resource destruction handler for it. Since you can store any kind of data as a resource, Zend needs to know how to free this resource if its not longer needed. This works by registering your own resource destruction handler to Zend which in turn gets called by Zend whenever your resource can be freed (whether manually or automatically). Registering your resource handler within Zend returns you the **resource type handle** for that resource. This handle is needed whenever you want to access a resource of this type later and is most of time stored in a global static variable within your extension. There is no need to worry about thread safety here because you only register your resource handler once during module initialization.

The Zend function to register your resource handler is defined as:

```
ZEND_API int zend_register_list_destructors_ex(rsrc_dtor_func_t ld, rsrc_dtor_func_t pld, ch
```

There are two different kinds of resource destruction handlers you can pass to this function: a handler for normal resources and a handler for persistent resources. Persistent resources are for example used for database connection. When registering a resource, either of these handlers must be given. For the other handler just pass NULL.

zend_register_list_destructors_ex() accepts the following parameters:

<code>ld</code>	Normal resource destruction handler callback
<code>pld</code>	Persistent resource destruction handler callback
<code>type_name</code>	A string specifying the name of your resource. It's always a good thing to specify a unique name within
<code>module_number</code>	The <code>module_number</code> is automatically available in your <code>PHP_MINIT_FUNCTION</code> function and therefore y

The return value is an unique integer ID for your **resource type**.

The resource destruction handler (either normal or persistent resources) has the following prototype:

```
void resource_destruction_handler(zend_rsrc_list_entry *rsrc TSRMLS_DC);
```

The passed `rsrc` is a pointer to the following structure:

```
typedef struct _zend_rsrc_list_entry {
    void *ptr;
    int type;
    int refcount;
};
```

```
} zend_rsrc_list_entry;
```

The member `void *ptr` is the actual pointer to your resource.

Now we know how to start things, we define our own resource we want register within Zend. It is only a simple structure with two integer members:

```
typedef struct {
    int resource_link;
    int resource_type;
} my_resource;
```

Our resource destruction handler is probably going to look something like this:

```
void my_destruction_handler(zend_rsrc_list_entry *rsrc TSRMLS_DC) {
    // You most likely cast the void pointer to your structure type
    my_resource *my_rsrc = (my_resource *) rsrc->ptr;

    // Now do whatever needs to be done with you resource. Closing
    // Files, Sockets, freeing additional memory, etc.
    // Also, don't forget to actually free the memory for your resource too!

    do_whatever_needs_to_be_done_with_the_resource(my_rsrc);
}
```

Note: One important thing to mention: If your resource is a rather complex structure which also contains pointers to memory you allocated during runtime you have to free them **before** freeing the resource itself!

Now that we have defined

1. what our resource is and
2. our resource destruction handler

we can go on and do the rest of the steps:

1. create a global variable within the extension holding the resource ID so it can be accessed from every function which needs it
2. define the resource name
3. write the resource destruction handler

4. and finally register the handler

```

// Somewhere in your extension, define the variable for your registered resources.
// If you wondered what 'le' stands for: it simply means 'list entry'.
static int le_myresource;

// It's nice to define your resource name somewhere
#define le_myresource_name "My type of resource"

[...]

// Now actually define our resource destruction handler
void my_destruction_handler(zend_rsrc_list_entry *rsrc TSRMLS_DC) {

    my_resource *my_rsrc = (my_resource *) rsrc->ptr;
    do_whatever_needs_to_be_done_with_the_resource(my_rsrc);
}

[...]

PHP_MINIT_FUNCTION(my_extension) {

    // Note that 'module_number' is already provided through the
    // PHP_MINIT_FUNCTION() function definition.

    le_myresource = zend_register_resource_destructors_ex(my_destruction_handler, NULL,

    // You can register additional resources, initialize
    // your global vars, constants, whatever.
}

```

To actually register a new resource you use can either use the **zend_register_resource()** function or the **ZEND_REGISTER_RESOURCE()** macro, both defined in `zend_list.h`. Although the arguments for both map 1:1 it's a good idea to always use macros to be upwards compatible:

```
int ZEND_REGISTER_RESOURCE(zval *rsrc_result, void *rsrc_pointer, int rsrc_type);
```

<code>rsrc_result</code>	This is an already initialized <code>zval *</code> container.
<code>rsrc_pointer</code>	Your resource pointer you want to store.
<code>rsrc_type</code>	The type which you received when you registered the resource destruction handler. If you followed the name

The return value is an unique integer identifier for that resource.

What is really going on when you register a new resource is it gets inserted in an internal list in Zend and the result is just stored in the given `zval *` container:

```
rsrc_id = zend_list_insert(rsrc_pointer, rsrc_type);
```

```

if (rsrc_result) {
    rsrc_result->value.lval = rsrc_id;
    rsrc_result->type = IS_RESOURCE;
}

return rsrc_id;

```

The returned `rsrc_id` uniquely identifies the newly registered resource. You can use the macro `RETURN_RESOURCE` to return it to the user:

```
RETURN_RESOURCE(rsrc_id)
```

Note: It is common practice that if you want to return the resource immediately to the user you specify the `return_value` as the `zval *` container.

Zend now keeps track of all references to this resource. As soon as all references to the resource are lost, the destructor that you previously registered for this resource is called. The nice thing about this setup is that you don't have to worry about memory leakages introduced by allocations in your module - just register all memory allocations that your calling script will refer to as resources. As soon as the script decides it doesn't need them anymore, Zend will find out and tell you.

Now that the user got his resource, at some point he is passing it back to one of your functions. The `value.lval` inside the `zval *` container contains the key to your resource and thus can be used to fetch the resource with the following macro: `ZEND_FETCH_RESOURCE`:

```
ZEND_FETCH_RESOURCE(rsrc, rsrc_type, rsrc_id, default_rsrc_id, resource_type_name, resource_
```

<code>rsrc</code>	This is your pointer which will point to your previously registered resource.
<code>rsrc_type</code>	This is the typecast argument for your pointer, e.g. <code>myresource *</code> .
<code>rsrc_id</code>	This is the address of the <code>zval *</code> container the user passed to your function, e.g. <code>&z_resource</code> if
<code>default_rsrc_id</code>	This integer specifies the default resource ID if no resource could be fetched or -1.
<code>resource_type_name</code>	This is the name of the requested resource. It's a string and is used when the resource can't be four
<code>resource_type</code>	The <code>resource_type</code> you got back when registering the resource destruction handler. In our exam

This macro has no return value. It is for the developers convenience and takes care of TSRMLS arguments passing and also does check if the resource could be fetched. It throws a warning message and returns the current PHP function with `NULL` if there was a problem retrieving the resource.

To force removal of a resource from the list, use the function `zend_list_delete()`. You can also force the reference count to increase if you know that you're creating another reference for a previously allocated value (for example, if you're automatically reusing a default database link). For this case, use the function `zend_list_addref()`. To search for previously allocated resource entries, use `zend_list_find()`. The complete API can be found in `zend_list.h`.

Macros for Automatic Global Variable Creation

In addition to the macros discussed earlier, a few macros allow easy creation of simple global variables. These are nice to know in case you want to introduce global flags, for example. This is somewhat bad practice, but Table 33-5 describes macros that do exactly this task. They don't need any `zval` allocation; you simply have to supply a variable name and value.

Table 33-5. Macros for Global Variable Creation

Macro	Description
<code>SET_VAR_STRING(name, value)</code>	Creates a new string.
<code>SET_VAR_STRINGL(name, value, length)</code>	Creates a new string of the specified length. This macro is faster than <code>SET_VAR_STRING</code> and also binary-safe.
<code>SET_VAR_LONG(name, value)</code>	Creates a new long.
<code>SET_VAR_DOUBLE(name, value)</code>	Creates a new double.

Creating Constants

Zend supports the creation of true constants (as opposed to regular variables). Constants are accessed without the typical dollar sign (\$) prefix and are available in all scopes. Examples include `TRUE` and `FALSE`, to name just two.

To create your own constants, you can use the macros in Table 33-6. All the macros create a constant with the specified name and value.

You can also specify flags for each constant:

- `CONST_CS` - This constant's name is to be treated as case sensitive.
- `CONST_PERSISTENT` - This constant is persistent and won't be "forgotten" when the current process carrying this constant shuts down.

To use the flags, combine them using a binary OR:

```
// register a new constant of type "long"
REGISTER_LONG_CONSTANT("NEW_MEANINGFUL_CONSTANT", 324, CONST_CS |
CONST_PERSISTENT);
```

There are two types of macros - `REGISTER_*_CONSTANT` and `REGISTER_MAIN_*_CONSTANT`. The first type creates constants bound to the current module. These constants are dumped from the symbol table as soon as the module that registered the constant is unloaded from memory. The second type creates constants that remain in the symbol table independently of the module.

Table 33-6. Macros for Creating Constants

Macro
<code>REGISTER_LONG_CONSTANT(name, value, flags)</code> <code>REGISTER_MAIN_LONG_CONSTANT(name, value, flags)</code>

REGISTER_DOUBLE_CONSTANT(name, value, flags)	REGISTER_MAIN_DOUBLE_CONSTANT(name, value, flags)
REGISTER_STRING_CONSTANT(name, value, flags)	REGISTER_MAIN_STRING_CONSTANT(name, value, flags)
REGISTER_STRINGL_CONSTANT(name, value, length, flags)	REGISTER_MAIN_STRINGL_CONSTANT(name, value, length, flags)

Chapter 34. Duplicating Variable Contents: The Copy Constructor

Sooner or later, you may need to assign the contents of one zval container to another. This is easier said than done, since the zval container doesn't contain only type information, but also references to places in Zend's internal data. For example, depending on their size, arrays and objects may be nested with lots of hash table entries. By assigning one zval to another, you avoid duplicating the hash table entries, using only a reference to them (at most).

To copy this complex kind of data, use the *copy constructor*. Copy constructors are typically defined in languages that support operator overloading, with the express purpose of copying complex types. If you define an object in such a language, you have the possibility of overloading the "=" operator, which is usually responsible for assigning the contents of the lvalue (result of the evaluation of the left side of the operator) to the rvalue (same for the right side).

Overloading means assigning a different meaning to this operator, and is usually used to assign a function call to an operator. Whenever this operator would be used on such an object in a program, this function would be called with the lvalue and rvalue as parameters. Equipped with that information, it can perform the operation it intends the "=" operator to have (usually an extended form of copying).

This same form of "extended copying" is also necessary for PHP's zval containers. Again, in the case of an array, this extended copying would imply re-creation of all hash table entries relating to this array. For strings, proper memory allocation would have to be assured, and so on.

Zend ships with such a function, called **zend_copy_ctor()** (the previous PHP equivalent was **pval_copy_constructor()**).

A most useful demonstration is a function that accepts a complex type as argument, modifies it, and then returns the argument:

```
zval *parameter;

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

// do modifications to the parameter here

// now we want to return the modified container:
*return_value == *parameter;
zval_copy_ctor(return_value);
```

The first part of the function is plain-vanilla argument retrieval. After the (left out) modifications, however, it gets interesting: The container of parameter is assigned to the (predefined) return_value container. Now, in order to effectively duplicate its contents, the copy constructor is called. The copy constructor works directly with the supplied argument, and the standard return values are FAILURE on failure and SUCCESS on success.

If you omit the call to the copy constructor in this example, both parameter and return_value would point to the same internal data, meaning that return_value would be an illegal additional reference to the same data structures. Whenever changes occurred in the data that parameter points to, return_value might be affected. Thus, in order to create separate copies, the copy constructor must be used.

The copy constructor's counterpart in the Zend API, the destructor **zval_dtor()**, does the opposite of the constructor.

Chapter 35. Returning Values

Returning values from your functions to PHP was described briefly in an earlier section; this section gives the details. Return values are passed via the `return_value` variable, which is passed to your functions as argument. The `return_value` argument consists of a `zval` container (see the earlier discussion of the call interface) that you can freely modify. The container itself is already allocated, so you don't have to run `MAKE_STD_ZVAL` on it. Instead, you can access its members directly.

To make returning values from functions easier and to prevent hassles with accessing the internal structures of the `zval` container, a set of predefined macros is available (as usual). These macros automatically set the correspondent type and value, as described in Table 35-1 and Table 35-2.

Note: The macros in Table 35-1 automatically *return* from your function, those in Table 35-2 only *set* the return value; they don't return from your function.

Table 35-1. Predefined Macros for Returning Values from a Function

Macro	Description
<code>RETURN_RESOURCE(resource)</code>	Returns a resource.
<code>RETURN_BOOL(bool)</code>	Returns a Boolean.
<code>RETURN_NULL()</code>	Returns nothing (a NULL value).
<code>RETURN_LONG(long)</code>	Returns a long.
<code>RETURN_DOUBLE(double)</code>	Returns a double.
<code>RETURN_STRING(string, duplicate)</code>	Returns a string. The duplicate flag indicates whether the string should be duplicated using <code>estrdup()</code> .
<code>RETURN_STRINGL(string, length, duplicate)</code>	Returns a string of the specified length; otherwise, behaves like <code>RETURN_STRING</code> . This macro is faster and binary-safe, however.
<code>RETURN_EMPTY_STRING()</code>	Returns an empty string.
<code>RETURN_FALSE</code>	Returns Boolean false.
<code>RETURN_TRUE</code>	Returns Boolean true.

Table 35-2. Predefined Macros for Setting the Return Value of a Function

Macro	Description
<code>RETVAL_RESOURCE(resource)</code>	Sets the return value to the specified resource.
<code>RETVAL_BOOL(bool)</code>	Sets the return value to the specified Boolean value.
<code>RETVAL_NULL</code>	Sets the return value to NULL.
<code>RETVAL_LONG(long)</code>	Sets the return value to the specified long.
<code>RETVAL_DOUBLE(double)</code>	Sets the return value to the specified double.
<code>RETVAL_STRING(string, duplicate)</code>	Sets the return value to the specified string and duplicates it to Zend internal memory if desired (see also <code>RETURN_STRING</code>).

<code>RETVAL_STRINGL(string, length, duplicate)</code>	Sets the return value to the specified string and forces the length to become length (see also <code>RETVAL_STRING</code>). This macro is faster and binary-safe, and should be used whenever the string length is known.
<code>RETVAL_EMPTY_STRING</code>	Sets the return value to an empty string.
<code>RETVAL_FALSE</code>	Sets the return value to Boolean false.
<code>RETVAL_TRUE</code>	Sets the return value to Boolean true.

Complex types such as arrays and objects can be returned by using `array_init()` and `object_init()`, as well as the corresponding hash functions on `return_value`. Since these types cannot be constructed of trivial information, there are no predefined macros for them.

Chapter 36. Printing Information

Often it's necessary to print messages to the output stream from your module, just as `print()` would be used within a script. PHP offers functions for most generic tasks, such as printing warning messages, generating output for `phpinfo()`, and so on. The following sections provide more details. Examples of these functions can be found on the CD-ROM.

zend_printf()

`zend_printf()` works like the standard `printf()`, except that it prints to Zend's output stream.

zend_error()

`zend_error()` can be used to generate error messages. This function accepts two arguments; the first is the error type (see `zend_errors.h`), and the second is the error message.

```
zend_error(E_WARNING, "This function has been called with empty arguments");
```

Table 36-1 shows a list of possible values (see Figure 36-1). These values are also referred to in `php.ini`. Depending on which error type you choose, your messages will be logged.

Table 36-1. Zend's Predefined Error Messages.

Error	Description
<code>E_ERROR</code>	Signals an error and terminates execution of the script immediately .
<code>E_WARNING</code>	Signals a generic warning. Execution continues.
<code>E_PARSE</code>	Signals a parser error. Execution continues.
<code>E_NOTICE</code>	Signals a notice. Execution continues. Note that by default the display of this type of error message
<code>E_CORE_ERROR</code>	Internal error by the core; shouldn't be used by user-written modules.
<code>E_COMPILE_ERROR</code>	Internal error by the compiler; shouldn't be used by user-written modules.
<code>E_COMPILE_WARNING</code>	Internal warning by the compiler; shouldn't be used by user-written modules.

Figure 36-1. Display of warning messages in the browser.

Including Output in `phpinfo()`

After creating a real module, you'll want to show information about the module in `phpinfo()` (in addition to the module name, which appears in the module list by default). PHP allows you to create your own section in the `phpinfo()` output with the `ZEND_MINFO()` function. This function should be placed in the module descriptor block (discussed earlier) and is always called whenever a script calls `phpinfo()`.

PHP automatically prints a section in `phpinfo()` for you if you specify the `ZEND_MINFO` function, including the module name in the heading. Everything else must be formatted and printed by you.

Typically, you can print an HTML table header using `php_info_print_table_start()` and then use the standard functions `php_info_print_table_header()` and `php_info_print_table_row()`. As arguments, both take the number of columns (as integers) and the column contents (as strings). Example 36-1 shows a source example and its output. To print the table footer, use `php_info_print_table_end()`.

Example 36-1. Source code and screenshot for output in `phpinfo()`.

```
php_info_print_table_start();
php_info_print_table_header(2, "First column", "Second column");
php_info_print_table_row(2, "Entry in first row", "Another entry");
php_info_print_table_row(2, "Just to fill", "another row here");
php_info_print_table_end();
```

Execution Information

You can also print execution information, such as the current file being executed. The name of the function currently being executed can be retrieved using the function `get_active_function_name()`. This function returns a pointer to the function name and doesn't accept any arguments. To retrieve the name of the file currently being executed, use `zend_get_executed_filename()`. This function accesses the executor globals, which are passed to it using the `TSRMLS_C` macro. The executor globals are automatically available to every function that's called directly by Zend (they're part of the `INTERNAL_FUNCTION_PARAMETERS` described earlier in this chapter). If you want to access the executor globals in another function that doesn't have them available automatically, call the macro `TSRMLS_FETCH()` once in that function; this will introduce them to your local scope.

Finally, the line number currently being executed can be retrieved using the function `zend_get_executed_lineno()`. This function also requires the executor globals as arguments. For examples of these functions, see Example 36-2.

Example 36-2. Printing execution information.

```
zend_printf("The name of the current function is %s<br>", get_active_function_name(TSRMLS_C));  
zend_printf("The file currently executed is %s<br>", zend_get_executed_filename(TSRMLS_C));  
zend_printf("The current line being executed is %i<br>", zend_get_executed_lineno(TSRMLS_C));
```

Chapter 37. Startup and Shutdown Functions

Startup and shutdown functions can be used for one-time initialization and deinitialization of your modules. As discussed earlier in this chapter (see the description of the Zend module descriptor block), there are global, module, and request startup and shutdown events.

The global startup functions are called once when PHP starts up; similarly, the global shutdown functions are called once when PHP shuts down. Please note that they're really only called *once*, not when a new Apache process is being created!

The module startup and shutdown functions are called whenever a module is loaded and needs initialization; the request startup and shutdown functions are called every time a request is processed (meaning that a file is being executed).

For dynamic extensions, module and request startup/shutdown events happen at the same time.

Declaration and implementation of these functions can be done with macros; see the earlier section "Declaration of the Zend Module Block" for details.

Chapter 38. Calling User Functions

You can call user functions from your own modules, which is very handy when implementing callbacks; for example, for array walking, searching, or simply for event-based programs.

User functions can be called with the function **call_user_function_ex()**. It requires a hash value for the function table you want to access, a pointer to an object (if you want to call a method), the function name, return value, number of arguments, argument array, and a flag indicating whether you want to perform zval separation.

```
ZEND_API int call_user_function_ex(HashTable *function_table, zval *object,
zval *function_name, zval **retval_ptr_ptr,
int param_count, zval **params[],
int no_separation);
```

Note that you don't have to specify both `function_table` and `object`; either will do. If you want to call a method, you have to supply the object that contains this method, in which case **call_user_function()** automatically sets the function table to this object's function table. Otherwise, you only need to specify `function_table` and can set `object` to `NULL`.

Usually, the default function table is the "root" function table containing all function entries. This function table is part of the compiler globals and can be accessed using the macro `CG`. To introduce the compiler globals to your function, call the macro `TSRMLS_FETCH` once.

The function name is specified in a zval container. This might be a bit surprising at first, but is quite a logical step, since most of the time you'll accept function names as parameters from calling functions within your script, which in turn are contained in zval containers again. Thus, you only have to pass your arguments through to this function. This zval must be of type `IS_STRING`.

The next argument consists of a pointer to the return value. You don't have to allocate memory for this container; the function will do so by itself. However, you have to destroy this container (using **zval_dtor()**) afterward!

Next is the parameter count as integer and an array containing all necessary parameters. The last argument specifies whether the function should perform zval separation - this should always be set to 0. If set to 1, the function consumes less memory but fails if any of the parameters need separation.

Example 38-1 shows a small demonstration of calling a user function. The code calls a function that's supplied to it as argument and directly passes this function's return value through as its own return value. Note the use of the constructor and destructor calls at the end - it might not be necessary to do it this way here (since they should be separate values, the assignment might be safe), but this is bulletproof.

Example 38-1. Calling user functions.

```
zval **function_name;
zval *retval;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &function_name) != SUCCESS))
{
    WRONG_PARAM_COUNT;
}

if((*function_name)->type != IS_STRING)
{
    zend_error(E_ERROR, "Function requires string argument");
}
```

```

}

TSRMLS_FETCH();

if(call_user_function_ex(CG(function_table), NULL, *function_name, &retval, 0, NULL, 0) != S
{
    zend_error(E_ERROR, "Function call failed");
}

zend_printf("We have %i as type<br>", retval->type);

*return_value = *retval;
zval_copy_ctor(return_value);
zval_ptr_dtor(&retval);

<?php

dl("call_userland.so");

function test_function()
{
    print("We are in the test function!<br>");
    return("hello");
}

$return_value = call_userland("test_function");

print("Return value: \"\$return_value\"<br>");
?>

```

Chapter 39. Initialization File Support

PHP 4 features a redesigned initialization file support. It's now possible to specify default initialization entries directly in your code, read and change these values at runtime, and create message handlers for change notifications.

To create an .ini section in your own module, use the macros `PHP_INI_BEGIN()` to mark the beginning of such a section and `PHP_INI_END()` to mark its end. In between you can use `PHP_INI_ENTRY()` to create entries.

```
PHP_INI_BEGIN()
PHP_INI_ENTRY("first_ini_entry", "has_string_value", PHP_INI_ALL, NULL)
PHP_INI_ENTRY("second_ini_entry", "2", PHP_INI_SYSTEM, OnChangeSecond)
PHP_INI_ENTRY("third_ini_entry", "xyz", PHP_INI_USER, NULL)
PHP_INI_END()
```

The `PHP_INI_ENTRY()` macro accepts four parameters: the entry name, the entry value, its change permissions, and a pointer to a change-notification handler. Both entry name and value must be specified as strings, regardless of whether they really are strings or integers.

The permissions are grouped into three sections: `PHP_INI_SYSTEM` allows a change only directly in the `php3.ini` file; `PHP_INI_USER` allows a change to be overridden by a user at runtime using additional configuration files, such as `.htaccess`; and `PHP_INI_ALL` allows changes to be made without restrictions. There's also a fourth level, `PHP_INI_PERDIR`, for which we couldn't verify its behavior yet.

The fourth parameter consists of a pointer to a change-notification handler. Whenever one of these initialization entries is changed, this handler is called. Such a handler can be declared using the `PHP_INI_MH` macro:

```
PHP_INI_MH(OnChangeSecond); // handler for ini-entry "second_ini_entry"

// specify ini-entries here

PHP_INI_MH(OnChangeSecond)
{
    zend_printf("Message caught, our ini entry has been changed to %s<br>", new_value);

    return(SUCCESS);
}
```

The new value is given to the change handler as string in the variable `new_value`. When looking at the definition of `PHP_INI_MH`, you actually have a few parameters to use:

```
#define PHP_INI_MH(name) int name(PHP_INI_ENTRY *entry, char *new_value,
                                uint new_value_length, void *mh_arg1,
                                void *mh_arg2, void *mh_arg3)
```

All these definitions can be found in `php_ini.h`. Your message handler will have access to a structure that contains the full entry, the new value, its length, and three optional arguments. These optional arguments can be specified with the additional macros `PHP_INI_ENTRY1` (allowing one additional

argument), `PHP_INI_ENTRY2` (allowing two additional arguments), and `PHP_INI_ENTRY3` (allowing three additional arguments).

The change-notification handlers should be used to cache initialization entries locally for faster access or to perform certain tasks that are required if a value changes. For example, if a constant connection to a certain host is required by a module and someone changes the hostname, automatically terminate the old connection and attempt a new one.

Access to initialization entries can also be handled with the macros shown in Table 39-1.

Table 39-1. Macros to Access Initialization Entries in PHP

Macro	Description
<code>INI_INT(name)</code>	Returns the current value of entry <code>name</code> as integer (long).
<code>INI_FLT(name)</code>	Returns the current value of entry <code>name</code> as float (double).
<code>INI_STR(name)</code>	Returns the current value of entry <code>name</code> as string. <i>Note:</i> This string is not duplicated, but instead points to the original string.
<code>INI_BOOL(name)</code>	Returns the current value of entry <code>name</code> as Boolean (defined as <code>zend_bool</code> , which currently means 0 or 1).
<code>INI_ORIG_INT(name)</code>	Returns the original value of entry <code>name</code> as integer (long).
<code>INI_ORIG_FLT(name)</code>	Returns the original value of entry <code>name</code> as float (double).
<code>INI_ORIG_STR(name)</code>	Returns the original value of entry <code>name</code> as string. <i>Note:</i> This string is not duplicated, but instead points to the original string.
<code>INI_ORIG_BOOL(name)</code>	Returns the original value of entry <code>name</code> as Boolean (defined as <code>zend_bool</code> , which currently means 0 or 1).

Finally, you have to introduce your initialization entries to PHP. This can be done in the module startup and shutdown functions, using the macros `REGISTER_INI_ENTRIES()` and

```
UNREGISTER_INI_ENTRIES():
```

```
ZEND_MINIT_FUNCTION(myModule)
{
    REGISTER_INI_ENTRIES();
}

ZEND_MSHUTDOWN_FUNCTION(myModule)
{
    UNREGISTER_INI_ENTRIES();
}
```

Chapter 40. Where to Go from Here

You've learned a lot about PHP. You now know how to create dynamic loadable modules and statically linked extensions. You've learned how PHP and Zend deal with internal storage of variables and how you can create and access these variables. You know quite a set of tool functions that do a lot of routine tasks such as printing informational texts, automatically introducing variables to the symbol table, and so on.

Even though this chapter often had a mostly "referential" character, we hope that it gave you insight on how to start writing your own extensions. For the sake of space, we had to leave out a lot; we suggest that you take the time to study the header files and some modules (especially the ones in the `ext/standard` directory and the MySQL module, as these implement commonly known functionality). This will give you an idea of how other people have used the API functions - particularly those that didn't make it into this chapter.

Chapter 41. Reference: Some Configuration Macros

config.m4

The file `config.m4` is processed by `buildconf` and must contain all the instructions to be executed during configuration. For example, these can include tests for required external files, such as header files, libraries, and so on. PHP defines a set of macros that can be used in this process, the most useful of which are described in Table 41-1.

Table 41-1. M4 Macros for `config.m4`

Macro	Description
<code>AC_MSG_CHECKING(message)</code>	Prints a "checking <message>"
<code>AC_MSG_RESULT(value)</code>	Gives the result to <code>AC_MSG_C</code>
<code>AC_MSG_ERROR(message)</code>	Prints message as error messa
<code>AC_DEFINE(name,value,description)</code>	Adds <code>#define</code> to <code>php_conf</code>
<code>AC_ADD_INCLUDE(path)</code>	Adds a compiler include path
<code>AC_ADD_LIBRARY_WITH_PATH(libraryname,librarypath)</code>	Specifies an additional library
<code>AC_ARG_WITH(modulename,description,unconditionaltest,conditionaltest)</code>	Quite a powerful macro, addi
<code>PHP_EXTENSION(modulename, [shared])</code>	This macro is a <i>must</i> to call fo

Chapter 42. API Macros

A set of macros was introduced into Zend's API that simplify access to zval containers (see Table 42-1).

Table 42-1. API Macros for Accessing zval Containers

Macro	Refers to
Z_LVAL(zval)	(zval).value.lval
Z_DVAL(zval)	(zval).value.dval
Z_STRVAL(zval)	(zval).value.str.val
Z_STRLEN(zval)	(zval).value.str.len
Z_ARRVAL(zval)	(zval).value.ht
Z_LVAL_P(zval)	(*zval).value.lval
Z_DVAL_P(zval)	(*zval).value.dval
Z_STRVAL_P(zval_p)	(*zval).value.str.val
Z_STRLEN_P(zval_p)	(*zval).value.str.len
Z_ARRVAL_P(zval_p)	(*zval).value.ht
Z_LVAL_PP(zval_pp)	(**zval).value.lval
Z_DVAL_PP(zval_pp)	(**zval).value.dval
Z_STRVAL_PP(zval_pp)	(**zval).value.str.val
Z_STRLEN_PP(zval_pp)	(**zval).value.str.len
Z_ARRVAL_PP(zval_pp)	(**zval).value.ht

Part VI. FAQ: Frequently Asked Questions

Chapter 43. General Information

This section holds the most general questions about PHP: what it is and what it does.

1. What is PHP?

From the preface of the manual:

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

A nice introduction to PHP by Stig Sæther Bakken can be found here (<http://www.zend.com/zend/art/intro.php>) on the Zend website. Also, much of the PHP Conference Material (<http://conf.php.net/>) is freely available.

2. What does PHP stand for?

PHP stands for *PHP: Hypertext Preprocessor*. This confuses many people because the first word of the acronym is the acronym. This type of acronym is called a recursive acronym. The curious can visit Free On-Line Dictionary of Computing (<http://www.foldoc.org/>) for more information on recursive acronyms.

3. What is the relation between the versions?

PHP/FI 2.0 is an early and no longer supported version of PHP. PHP 3 is the successor to PHP/FI 2.0 and is a lot nicer. PHP 4 is the latest generation of PHP, which uses the Zend engine (<http://www.zend.com/>) under the hood.

4. Can I run several versions of PHP at the same time?

Yes. See the `INSTALL` file that is included in the PHP 4 source distribution. Also, read the related appendix.

5. What are the differences between PHP 3 and PHP 4?

There are a couple of articles (<http://www.zend.com/zend/art/>) written on this by the authors of PHP 4. Here's a list of some of the more important new features:

- Extended API module
- Generalized build process under UNIX
- Generic web server interface that also supports multi-threaded web servers
- Improved syntax highlighter
- Native HTTP session support
- Output buffering support
- More powerful configuration system
- Reference counting

Please see the What's new in PHP 4 overview (<http://www.zend.com/zend/whats-new.php>) for a detailed explanation of these features and more. If you're migrating from PHP 3 to PHP 4, also read the related appendix.

6. I think I found a bug! Who should I tell?

You should go to the PHP Bug Database and make sure the bug isn't a known bug. If you don't see it in the database, use the reporting form to report the bug. It is important to use the bug database instead of just sending an email to one of the mailing lists because the bug will have a tracking number assigned and it will then be possible for you to go back later and check on the status of the bug. The bug database can be found at <http://bugs.php.net/>.

Chapter 44. Mailing lists

This section holds questions about how to get in touch with the PHP community. The best way is the mailing lists.

1. Are there any PHP mailing lists?

Of course! There are many mailing lists for several subjects. A whole list of mailing lists can be found on our Support (<http://www.php.net/support.php>) page.

The most general mailing list is `php-general`. To subscribe, send mail to `php-general-subscribe@lists.php.net` (`mailto:php-general-subscribe@lists.php.net`). You don't need to include anything special in the subject or body of the message. To unsubscribe, send mail to `php-general-unsubscribe@lists.php.net` (`mailto:php-general-unsubscribe@lists.php.net`).

You can also subscribe and unsubscribe using the web interface on our Support (<http://www.php.net/support.php>) page.

2. Are there any other communities?

There are countless of them around the world. We have links for example to some IRC servers and foreign language mailing lists on our Support (<http://www.php.net/support.php>) page.

3. Help! I can't seem to subscribe/unsubscribe to/from one of the mailing lists!

If you have problems subscribing to or unsubscribing from the `php-general` mailing list, it may be because the mailing list software can't figure out the correct mailing address to use. If your email address was `joeblow@example.com`, you can send your subscription request to `php-general-subscribe-joeblow@example.com@lists.php.net`, or your unsubscription request to `php-general-unsubscribe-joeblow@example.com@lists.php.net`. Use similar addresses for the other mailing lists.

4. Is there an archive of the mailing lists anywhere?

Yes, you will find a list of archive sites on the Support (<http://www.php.net/support.php>) page. The mailing list articles are also archived as news messages. You can access the news server at `news://news.php.net/` with a news client. There is also an experimental web interface for the news server at <http://news.php.net/>

5. What can I ask the mailing list?

Since PHP is growing more and more popular by the day the traffic has increased on the `php-general` mailing list and as of now the list gets about 150 to 200 posts a day. Because of this it is in everyones interest that you use the list as a last resort when you have looked everywhere else.

Before you post to the list please have a look in this FAQ and the manual to see if you can find the help there. If there is nothing to be found there try out the mailing list archives (see above). If you're having problem with installing or configuring PHP please read through all included documentation and README's. If you still can't find any information that helps you out you're more than welcome to use the mailing list.

6. What information should I include when posting to the mailing list?

Posts like "I can't get PHP up and running! Help me! What is wrong?" are of absolutely no use to anyone. If you're having problems getting PHP up and running you must include what operating system you are running on, what version of PHP you're trying to set up, how you got it (pre-compiled, CVS, RPMs and so on), what you have done so far, where you got stuck and the exact error message.

This goes for any other problem as well. You have to include information on what you have done, where you got stuck, what you're trying to do and, if applicable, exact error messages. If you're having problems with your source code you need to include the part of the code that isn't working. Do not include more code than necessary though! It makes the post hard to read and a lot of people might just skip it all together because of this. If you're unsure about how much information to include in the mail it's better that you include too much than too little.

Another important thing to remember is to summarize your problem on the subject line. A subject like "HELP MEEEE!!!" or "What is the problem here?" will be ignored by the majority of the readers.

Chapter 45. Obtaining PHP