Installation and Configuration

General Installation Considerations

Before starting the installation, 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:

- Websites and web applications (server-side scripting)
- Command line scripting
- Desktop (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 (e.g. Apache on Linux and MacOS X; 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.

In case of 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 or FastCGI processor. This means you set up your server to use the CGI executable of PHP to process all PHP file requests on the server.

If you are also interested to use PHP for command line scripting (e.g. 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 desktop 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. 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. You will also find information on the command line executable in the following sections.

PHP source code and binary distributions for Windows can be found at » http://www.php.net/downloads.php. We recommend you to choose a » mirror nearest to you for downloading the distributions.

Installation on Unix systems

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

As our manual outlines in the General Installation Considerations section, we are mainly dealing with web centric setups of PHP in this section, although we will cover setting up PHP for command line usage as well.

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. Many Unix like systems have some sort of package installation system. 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 web server. 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.

Prerequisite knowledge and software for compiling:

- Basic Unix skills (being able to operate "make" and a C compiler)
- An ANSI C compiler
- flex: Version 2.5.4
- bison: Version 1.28 (preferred), 1.35, or 1.75
- A web server
- Any module specific components (such as gd, pdf libs, etc.)

The initial PHP setup and configuration process is controlled by the use of the command line options of the *configure* script. You could get a list of all available options along with short explanations running*./configure --help*. Our manual documents the different options separately. You will find the core options in the appendix, while the different extension specific options are descibed on the reference pages.

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

Apache 1.3.x on Unix systems

This section contains notes and hints specific to Apache installs of PHP on Unix platforms. We also have instructions and notes for Apache 2 on a separate page.

You can select arguments to add to the *configure* on line 10 below from the list of core configure options and from extension specific options described at the respective places in the manual. 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 #1 - Installation Instructions (Apache Shared Module Version) for PHP

```
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. Now, configure your PHP.  This is where you customize your PHP
   with various options, like which extensions will be enabled.  Do a
   ./configure --help for a list of available options.  In our example
   we'll do a simple configure with Apache 1 and MySQL support.  Your
   path to apxs may differ from our example.

     ./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.

   Note that unless told otherwise, 'make install' will also install PEAR,
   various PHP tools such as phpize, install the PHP CLI, and more.

13. Setup your php.ini file:

     cp php.ini-dist /usr/local/lib/php.ini

   You may edit your .ini file to set PHP options.  If you prefer your
   php.ini in another location, use --with-config-file-path=/some/path in
   step 10.

   If you instead choose php.ini-recommended, be certain to read the list
   of changes within, as they affect how PHP behaves.

14. Edit your httpd.conf to load the PHP module.  The path on the right hand
   side of the LoadModule statement must point to the path of the PHP
   module on your system.  The make install from above may have already
   added this for you, but be sure to check.

   For PHP 4:

     LoadModule php4_module libexec/libphp4.so

   For PHP 5:
     LoadModule php5_module libexec/libphp5.so

15. And in the AddModule section of httpd.conf, somewhere under the
   ClearModuleList, add this:

   For PHP 4:

     AddModule mod_php4.c

   For PHP 5:

     AddModule mod_php5.c

16. Tell Apache to parse certain extensions as PHP.  For example,
   let's have Apache parse the .php extension as PHP.  You could
   have any extension(s) parse as PHP by simply adding more, with
   each separated by a space.  We'll add .phtml to demonstrate.

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

   It's also common to setup the .phps extension to show highlighted PHP
   source, this can be done with:

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

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

Alternatively, to install PHP as a static object:

Example #2 - Installation Instructions (Static Module Installation for Apache) for PHP

```
1.  gunzip -c apache_1.3.x.tar.gz | tar xf -
2.  cd apache_1.3.x
3.  ./configure
4.  cd ..

5.  gunzip -c php-5.x.y.tar.gz | tar xf -
6.  cd php-5.x.y
7.  ./configure --with-mysql --with-apache=../apache_1.3.x
8.  make
9.  make install

10. cd ../apache_1.3.x

11. ./configure --prefix=/www --activate-module=src/modules/php5/libphp5.a
   (The above line is correct! Yes, we know libphp5.a does not exist at this
   stage. It isn't supposed to. It will be created.)

12. make
   (you should now have an httpd binary which you can copy to your Apache bin
dir if
   it is your first install then you need to "make install" as well)

13. cd ../php-5.x.y
```

```
14. cp php.ini-dist /usr/local/lib/php.ini

15. You can edit /usr/local/lib/php.ini file to set PHP options.
   Edit your httpd.conf or srm.conf file and add:
   AddType application/x-httpd-php .php
```

Note

Replace *php-5* by *php-4* and *php5* by *php4* in PHP 4.

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.

Example #3 - Example commands for restarting Apache

```
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 *libphp5.so* (or *libphp4.so* in PHP 4) shared library that is loaded into Apache using a LoadModule line in Apache's *httpd.conf* file. The PostgreSQL support is embedded into this 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 *libmodphp5.a* library, a *mod_php5.c* and some accompanying files and copy this into the *src/modules/php5* directory in the Apache source tree. Then you compile Apache using *--activate-module=src/modules/php5/libphp5.a* and the Apache build system will create *libphp5.a* and link it statically into the *httpd* binary (replace *php5* by *php4* in PHP 4). 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 » web page on DSO support.

Note

Apache's default *httpd.conf* currently ships with a section that looks like this:

```
User nobody
Group "#-1"
```

Unless you change that to "Group nogroup" or something like that ("Group daemon" is also very common) PHP will not be able to open files.

Note

Make sure you specify the installed version of apxs when using

--with-apxs=/path/to/apxs. You must NOT use the apxs version that is in the apache sources but the one that is actually installed on your system.

Apache 2.0 on Unix systems

This section contains notes and hints specific to Apache 2.0 installs of PHP on Unix systems.

Warning

We do not recommend using a threaded MPM in production with Apache 2. Use the prefork MPM instead, or use Apache 1. For information on why, read the related FAQ entry on using Apache2 with a threaded MPM

You are highly encouraged to take a look at the » Apache Documentation to get a basic understanding of the Apache 2.0 Server.

Note

PHP and Apache 2.0.x compatibility notes

The following versions of PHP are known to work with the most recent version of Apache 2.0.x:

- PHP 4.3.0 or later available at » http://www.php.net/downloads.php.
- the latest stable development version. Get the source code » http://snaps.php.net/php5-latest.tar.gz or download binaries for Windows » http://snaps.php.net/win32/php5-win32-latest.zip.
- a prerelease version downloadable from » http://qa.php.net/.
- you have always the option to obtain PHP through » anonymous CVS.

These versions of PHP are compatible to Apache 2.0.40 and later.

Apache 2.0 *SAPI* -support started with PHP 4.2.0. PHP 4.2.3 works with Apache 2.0.39, don't use any other version of Apache with PHP 4.2.3. However, the recommended setup is to use PHP 4.3.0 or later with the most recent version of Apache2.

All mentioned versions of PHP will work still with Apache 1.3.x.

Download the most recent version of » Apache 2.0 and a fitting PHP version from the above mentioned places. This quick guide covers only the basics to get started with Apache 2.0 and PHP. For more information read the » Apache Documentation. The version numbers have been omitted here, to ensure the instructions are not incorrect. You will need to replace the 'NN' here with the correct values from your files.

Example #4 - Installation Instructions (Apache 2 Shared Module Version)

```
1.  gzip -d httpd-2_0_NN.tar.gz
2.  tar xvf httpd-2_0_NN.tar
3.  gunzip php-NN.tar.gz
4.  tar -xvf php-NN.tar
5.  cd httpd-2_0_NN
6.  ./configure --enable-so
7.  make
8.  make install

   Now you have Apache 2.0.NN available under /usr/local/apache2,
   configured with loadable module support and the standard MPM prefork.
   To test the installation use your normal procedure for starting
   the Apache server, e.g.:
   /usr/local/apache2/bin/apachectl start
   and stop the server to go on with the configuration for PHP:
   /usr/local/apache2/bin/apachectl stop.

9.  cd ../php-NN

10. Now, configure your PHP.  This is where you customize your PHP
   with various options, like which extensions will be enabled.  Do a
   ./configure --help for a list of available options.  In our example
   we'll do a simple configure with Apache 2 and MySQL support.  Your
   path to apxs may differ, in fact, the binary may even be named apxs2 on
   your system.

     ./configure --with-apxs2=/usr/local/apache2/bin/apxs --with-mysql

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.

   Note that unless told otherwise, 'make install' will also install PEAR,
   various PHP tools such as phpize, install the PHP CLI, and more.

13. Setup your php.ini

   cp php.ini-dist /usr/local/lib/php.ini

   You may edit your .ini file to set PHP options.  If you prefer having
   php.ini in another location, use --with-config-file-path=/some/path in
   step 10.

   If you instead choose php.ini-recommended, be certain to read the list
   of changes within, as they affect how PHP behaves.

14. Edit your httpd.conf to load the PHP module.  The path on the right hand
   side of the LoadModule statement must point to the path of the PHP
   module on your system.  The make install from above may have already
   added this for you, but be sure to check.

   For PHP 4:
     LoadModule php4_module modules/libphp4.so

   For PHP 5:

     LoadModule php5_module modules/libphp5.so

15. Tell Apache to parse certain extensions as PHP.  For example, let's have
   Apache parse .php files as PHP.  Instead of only using the Apache AddType
   directive, we want to avoid potentially dangerous uploads and created
   files such as exploit.php.jpg from being executed as PHP.  Using this
   example, you could have any extension(s) parse as PHP by simply adding
   them.  We'll add .phtml to demonstrate.

     <FilesMatch \.php$>
         SetHandler application/x-httpd-php
     </FilesMatch>

   Or, if we wanted to allow .php, .php2, .php3, .php4, .php5, .php6, and
   .phtml files to be executed as PHP, but nothing else, we'd use this:

     <FilesMatch "\.ph(p[2-6]?|tml)$">
         SetHandler application/x-httpd-php
     </FilesMatch>

   And to allow .phps files to be executed as PHP source files, add this:

     <FilesMatch "\.phps$">
         SetHandler application/x-httpd-php-source
     </FilesMatch>

16. Use your normal procedure for starting the Apache server, e.g.:

     /usr/local/apache2/bin/apachectl start

         - OR -

     service httpd restart
```

Following the steps above you will have a running Apache2 web server with support for PHP as a *SAPI* module. Of course there are many more configuration options available Apache and PHP. For more information type*./configure --help* in the corresponding source tree. If you wish to build a multithreaded version of Apache2, you must overwrite the standard MPM-Module *prefork* either with *worker* or *perchild*. To do so append to your configure line in step 6 above either the option *--with-mpm=worker* or *--with-mpm=perchild*. Before doing so, please beware the consequences and have at least a fair understand of what the implications. For more information, read the Apache documentation regarding » MPM-Modules.

Note

If you want to use content negotiation, read the Apache MultiViews FAQ.

Note

To build a multithreaded version of Apache your system must support threads. This also implies to build PHP with experimental Zend Thread Safety (ZTS). Therefore not all extensions might be available. The recommended setup is to build Apache with the standard *prefork* MPM-Module.

Lighttpd 1.4 on Unix systems

This section contains notes and hints specific to Lighttpd 1.4 installs of PHP on Unix systems.

Please use the » Lighttpd trac to learn how to install Lighttpd properly before continuing.

Fastcgi is the preferred SAPI to connect PHP and Lighttpd. Fastcgi is automagically enabled in php-cgi in PHP5.3, but for older versions configure php with --enable-fastcgi. To confirm that PHP has fastcgi enabled, *php -v* should contain *PHP 5.2.5 (cgi-fcgi)* Before PHP 5.2.3, fastcgi was enabled on the php binary (there was no php-cgi).

Letting Lighttpd spawn php processes

To configure Lighttpd to connect to php and spawn fastcgi processes, edit lighttpd.conf. Sockets are preferred to connect to fastcgi processes on the local system.

Example #5 - Partial lighttpd.conf

```
server.modules += ( "mod_fastcgi" )

fastcgi.server = ( ".php" =>
 ((
   "socket" => "/tmp/php.socket",
   "bin-path" => "/usr/local/bin/php-cgi",
   "bin-environment" => (
     "PHP_FCGI_CHILDREN" => "16",
     "PHP_FCGI_MAX_REQUESTS" => "10000"
   )
   "min-procs" => 1,
   "max-procs" => 1,
   "idle-timeout" => 20
 ))
)
```

The bin-path directive allows lighttpd to spawn fastcgi processes dynamically. PHP will spawn children according to the PHP_FCGI_CHILDREN environment variable. The "bin-environment" directive sets the environment for the spawned processes. PHP will kill a child process after the number of requests specified by PHP_FCGI_MAX_REQUESTS is reached. The directives "min-procs" and "max-procs" should generally be avoided with PHP. PHP manages its own children and opcode caches like APC will only share among children managed by PHP. If "min-procs" is set to something greater than 1, the total number of php responders will be multiplied PHP_FCGI_CHILDREN (2 min-procs * 16 children gives 32 responders).

Spawning with spawn-fcgi

Lighttpd provides a program called spawn-fcgi to ease the process of spawning fastcgi processes easier.

Spawning php-cgi

It is possible to spawn processes without spawn-fcgi, though a bit of heavy-lifting is required. Setting the PHP_FCGI_CHILDREN environment var controls how many children PHP will spawn to handle incoming requests. Setting PHP_FCGI_MAX_REQUESTS will determine how long (in requests) each child will live. Here's a simple bash script to help spawn php responders.

Example #6 - Spawning FastCGI Responders

```
#!/bin/sh

# Location of the php-cgi binary
PHP=/usr/local/bin/php-cgi

# PID File location
PHP_PID=/tmp/php.pid

# Binding to an address
#FCGI_BIND_ADDRESS=10.0.1.1:10000
# Binding to a domain socket
FCGI_BIND_ADDRESS=/tmp/php.sock

PHP_FCGI_CHILDREN=16
PHP_FCGI_MAX_REQUESTS=10000

env -i PHP_FCGI_CHILDREN=$PHP_FCGI_CHILDREN \
      PHP_FCGI_MAX_REQUESTS=$PHP_FCGI_MAX_REQUESTS \
      $PHP -b $FCGI_BIND_ADDRESS &

echo $! > "$PHP_PID"
```

Connecting to remote FCGI instances

Fastcgi instances can be spawned on multiple remote machines in order to scale applications.

Example #7 - Connecting to remote php-fastcgi instances

```
fastcgi.server = ( ".php" =>
   (( "host" => "10.0.0.2", "port" => 1030 ),
    ( "host" => "10.0.0.3", "port" => 1030 ))
)
```

Caudium

PHP can be built as a Pike module for the » Caudium webserver. Follow the simple instructions below to install PHP for Caudium.

Example #8 - 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 reference pages for extension specific configure options.

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.

fhttpd related notes

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.

Note
Support for fhttpd is no longer available as of PHP 4.3.0.

Sun, iPlanet and Netscape servers on Sun Solaris

This section contains notes and hints specific to Sun Java System Web Server, Sun ONE Web Server, iPlanet and Netscape server installs of PHP on Sun Solaris.

From PHP 4.3.3 on you can use PHP scripts with the NSAPI module to generate custom directory listings and error pages. Additional functions for Apache compatibility are also available. For support in current web servers read the note about subrequests.

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

To build PHP with Sun JSWS/Sun ONE WS/iPlanet/Netscape 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*.

- Install the following packages from » http://www.sunfreeware.com/ or another download site:
  - *autoconf-2.13*
  - *automake-1.4*
  - *bison-1_25-sol26-sparc-local*
  - *flex-2_5_4a-sol26-sparc-local*
  - *gcc-2_95_2-sol26-sparc-local*
  - *gzip-1.2.4-sol26-sparc-local*
  - *m4-1_4-sol26-sparc-local*
  - *make-3_76_1-sol26-sparc-local*
  - *mysql-3.23.24-beta* (if you want mysql support)
  - *perl-5_005_03-sol26-sparc-local*
  - *tar-1.13* (GNU tar)
- Make sure your path includes the proper directories *PATH=.:/usr/local/bin:/usr/sbin:/usr/bin:/usr/ccs/bin* and make it available to your system **export PATH**.
- **gunzip php-x.x.x.tar.gz** (if you have a .gz dist, otherwise go to 4).
- **tar xvf php-x.x.x.tar**
- Change to your extracted PHP directory: **cd ../php-x.x.x**
- For the following step, make sure */opt/netscape/suitespot/* is where your netscape server is installed. Otherwise, change to the correct path and run:

```
./configure --with-mysql=/usr/local/mysql \
--with-nsapi=/opt/netscape/suitespot/ \
--enable-libgcc
```

- Run *make* followed by *make install*.

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

Configuration Instructions for Sun/iPlanet/Netscape

Firstly you may need to add some paths to the *LD_LIBRARY_PATH* environment for the server to find all the shared libs. This can best done in the start script for your web server. 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/*.

- Add the following line to *mime.types* (you can do that by the administration server):
  ```
  type=magnus-internal/x-httpd-php exts=php
  ```

- Edit *magnus.conf* (for servers >= 6) or *obj.conf* (for servers < 6) and add the following, shlib will vary depending on your system, it will be something like */opt/netscape/suitespot/bin/libphp4.so*. You should place the following lines after *mime types init*.
  ```
  Init fn="load-modules" funcs="php4_init,php4_execute,php4_auth_trans"
  shlib="/opt/netscape/suitespot/bin/libphp4.so"
  Init fn="php4_init" LateInit="yes" errorString="Failed to initialize PHP!"
  [php_ini="/path/to/php.ini"]
  ```
  (PHP >= 4.3.3) The *php_ini* parameter is optional but with it you can place your *php.ini* in your web server config directory.

- Configure the default object in *obj.conf* (for virtual server classes [version 6.0+] in their *vserver.obj.conf*):
  ```
  <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" [inikey=value
  inikey=value ...]
  .
  .
  </Object>
  ```
  (PHP >= 4.3.3) As additional parameters you can add some special *php.ini* -values, for example you can set a *docroot="/path/to/docroot"* specific to the context *php4_execute* is called. For boolean ini-keys please use 0/1 as value, not *"On","Off",...* (this will not work correctly), e.g. *zlib.output_compression=1* instead of *zlib.output_compression="On"*

- This is only needed if you want to configure a directory that only consists of PHP scripts (same like a *cgi-bin* directory):
  ```
  <Object name="x-httpd-php">
  ObjectType fn="force-type" type="magnus-internal/x-httpd-php"
  ```

```
Service fn=php4_execute [inikey=value inikey=value ...]
</Object>
```

After that you can configure a directory in the Administration server and assign it the style *x-httpd-php*. All files in it will get executed as PHP. This is nice to hide PHP usage by renaming files to*.html*.

- Setup of authentication: 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 to your default object:

  ```
  <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>
  ```

Note
The stacksize that PHP uses depends on the configuration of the web server. If you get crashes with very large PHP scripts, it is recommended to raise it with the Admin Server (in the section "MAGNUS EDITOR").

CGI environment and recommended modifications in php.ini

Important when writing PHP scripts is the fact that Sun JSWS/Sun ONE WS/iPlanet/Netscape is a multithreaded web server. Because of that all requests are running in the same process space (the space of the web server itself) and this space has only one environment. If you want to get CGI variables like *PATH_INFO*, *HTTP_HOST* etc. it is not the correct way to try this in the old PHP way with getenv() or a similar way (register globals to environment, *$_ENV*). You would only get the environment of the running web server without any valid CGI variables!

Note
Why are there (invalid) CGI variables in the environment? Answer: This is because you started the web server process from the admin server which runs the startup script of the web server, you wanted to start, as a CGI script (a CGI script inside of the admin server!). This is why the environment of the started web server has some CGI environment variables in it. You can test this by starting the web

server not from the administration server. Use the command line as root user and start it manually - you will see there are no CGI-like environment variables.

Simply change your scripts to get CGI variables in the correct way for PHP 4.x by using the superglobal *$_SERVER*. If you have older scripts which use *$HTTP_HOST*, etc., you should turn on *register_globals* in *php.ini* and change the variable order too (important: remove *"E"* from it, because you do not need the environment here):

```
variables_order = "GPCS"
register_globals = On
```

Special use for error pages or self-made directory listings (PHP >= 4.3.3)

You can use PHP to generate the error pages for *"404 Not Found"* or similar. Add the following line to the object in *obj.conf* for every error page you want to overwrite:

```
Error fn="php4_execute" code=XXX script="/path/to/script.php" [inikey=value
inikey=value...]
```

where *XXX* is the HTTP error code. Please delete any other *Error* directives which could interfere with yours. If you want to place a page for all errors that could exist, leave the *code* parameter out. Your script can get the HTTP status code with *$_SERVER['ERROR_TYPE']*.

Another possibility is to generate self-made directory listings. Just create a PHP script which displays a directory listing and replace the corresponding default Service line for *type="magnus-internal/directory"* in *obj.conf* with the following:

```
Service fn="php4_execute" type="magnus-internal/directory"
script="/path/to/script.php" [inikey=value inikey=value...]
```

For both error and directory listing pages the original URI and translated URI are in the variables *$_SERVER['PATH_INFO']* and *$_SERVER['PATH_TRANSLATED']*.

Note about nsapi_virtual() and subrequests (PHP >= 4.3.3)

The NSAPI module now supports the nsapi_virtual() function (alias: virtual()) to make subrequests on the web server and insert the result in the web page. This function uses some undocumented features from the NSAPI library. On Unix the module automatically looks for the needed functions and uses them if available. If not, nsapi_virtual() is disabled.

Note

But be warned: Support for nsapi_virtual() is EXPERIMENTAL!!!

CGI and command line setups

The default is to build PHP as a CGI program. This creates a command line 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 users to run different

PHP-enabled pages under different user-ids.

Warning
A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

As of PHP 4.3.0, some important additions have happened to PHP. A new SAPI named CLI also exists and it has the same name as the CGI binary. What is installed at *{PREFIX}/bin/php* depends on your configure line and this is described in detail in the manual section named Using PHP from the command line. For further details please read that section of the manual.

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.

Using Variables

Some server supplied environment variables are not defined in the current » CGI/1.1 specification. Only the following variables are defined there: *AUTH_TYPE*, *CONTENT_LENGTH*, *CONTENT_TYPE*, *GATEWAY_INTERFACE*, *PATH_INFO*, *PATH_TRANSLATED*, *QUERY_STRING*, *REMOTE_ADDR*, *REMOTE_HOST*, *REMOTE_IDENT*, *REMOTE_USER*, *REQUEST_METHOD*, *SCRIPT_NAME*, *SERVER_NAME*, *SERVER_PORT*, *SERVER_PROTOCOL*, and *SERVER_SOFTWARE*. Everything else should be treated as 'vendor extensions'.

HP-UX specific installation notes

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

There are two main options for installing PHP on HP-UX systems. Either compile it, or install a pre-compiled binary.

Official pre-compiled packages are located here: » http://software.hp.com/

Until this manual section is rewritten, the documentation about compiling PHP (and related extensions) on HP-UX systems has been removed. For now, consider reading the following external resource: » Building Apache and PHP on HP-UX 11.11

OpenBSD installation notes

This section contains notes and hints specific to installing PHP on » OpenBSD 3.6.

Using Binary Packages

Using binary packages to install PHP on OpenBSD is the recommended and simplest method. The core package has been separated from the various modules, and each can be installed and removed independently from the others. The files you need can be found on your OpenBSD CD or on the FTP site.

The main package you need to install is *php4-core-4.3.8.tgz*, which contains the basic engine (plus gettext and iconv). Next, take a look at the module packages, such as *php4-mysql-4.3.8.tgz* or *php4-imap-4.3.8.tgz*. You need to use the *phpxs* command to activate and deactivate these modules in your *php.ini*.

Example #9 - OpenBSD Package Install Example

```
# pkg_add php4-core-4.3.8.tgz
# /usr/local/sbin/phpxs -s
# cp /usr/local/share/doc/php4/php.ini-recommended /var/www/conf/php.ini
 (add in mysql)
# pkg_add php4-mysql-4.3.8.tgz
# /usr/local/sbin/phpxs -a mysql
 (add in imap)
# pkg_add php4-imap-4.3.8.tgz
# /usr/local/sbin/phpxs -a imap
 (remove mysql as a test)
# pkg_delete php4-mysql-4.3.8
# /usr/local/sbin/phpxs -r mysql
 (install the PEAR libraries)
# pkg_add php4-pear-4.3.8.tgz
```

Read the » packages(7) manual page for more information about binary packages on OpenBSD.

Using Ports

You can also compile up PHP from source using the » ports tree. However, this is only recommended for users familiar with OpenBSD. The PHP 4 port is split into two sub-directories: core and extensions. The extensions directory generates sub-packages for all of the supported PHP modules. If you find you do not want to create some of these modules, use the *no_** FLAVOR. For example, to skip building the imap module, set the FLAVOR to *no_imap*.

Common Problems

- The default install of Apache runs inside a » chroot(2) jail, which will restrict PHP scripts to accessing files under */var/www*. You will therefore need to create a */var/www/tmp* directory for PHP session files to be stored, or use an alternative session backend. In addition, database sockets need to be placed inside the jail or listen on the *localhost* interface. If you use network functions, some files from */etc* such as */etc/resolv.conf* and */etc/services* will need to be moved into */var/www/etc*. The

OpenBSD PEAR package automatically installs into the correct chroot directories, so no special modification is needed there. More information on the OpenBSD Apache is available in the » OpenBSD FAQ.

- The OpenBSD 3.6 package for the » gd extension requires XFree86 to be installed. If you do not wish to use some of the font features that require X11, install the *php4-gd-4.3.8-no_x11.tgz* package instead.

Older Releases

Older releases of OpenBSD used the FLAVORS system to compile up a statically linked PHP. Since it is hard to generate binary packages using this method, it is now deprecated. You can still use the old stable ports trees if you wish, but they are unsupported by the OpenBSD team. If you have any comments about this, the current maintainer for the port is Anil Madhavapeddy (avsm at openbsd dot org).

Solaris specific installation tips

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. Read this FAQ for information on why using GNU versions for some of these tools is necessary. The required software is as follows:

- gcc (recommended, other C compilers may work)
- make
- flex
- bison
- m4
- autoconf
- automake
- perl
- gzip
- tar
- GNU sed

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.

Debian GNU/Linux installation notes

This section contains notes and hints specific to installing PHP on [» Debian GNU/Linux](#).

Using APT

While you can just download the PHP source and compile it yourself, using Debian's packaging system is the simplest and cleanest method of installing PHP. If you are not familiar with building software on Linux, this is the way to go.

The first decision you need to make is whether you want to install Apache 1.3.x or Apache 2.x. The corresponding PHP packages are respectively named libapache-mod-php* and libapache2-mod-php*. The steps given below will use Apache 1.3.x. Please note that, as of this writing, there is no official Debian packages of PHP 5. Then the steps given below will install PHP 4.

PHP is available in Debian as CGI or CLI flavour too, named respectively php4-cgi and php4-cli. If you need them, you'll just have to reproduce the following steps with the good package names. Another special package you'd want to install is php4-pear. It contains a minimal PEAR installation and the *pear* commandline utility.

If you need more recent packages of PHP than the Debian's stable ones or if some PHP modules lacks the Debian official repository, perhaps you should take a look at » http://www.apt-get.org/. One of the results found should be » Dotdeb. This unofficial repository is maintained by » Guillaume Plessis and contains Debian packages of the most recent versions of PHP 4 and PHP 5. To use it, just add the to following two lines to your */etc/apt/sources.lists* and run *apt-get update*:

Example #10 - The two Dotdeb related lines

```
deb http://packages.dotdeb.org stable all
deb-src http://packages.dotdeb.org stable all
```

The last thing to consider is whether your list of packages is up to date. If you have not updated it recently, you need to run *apt-get update* before anything else. This way, you will be using the most recent stable version of the Apache and PHP packages.

Now that everything is in place, you can use the following example to install Apache and PHP:

Example #11 - Debian Install Example with Apache 1.3

```
# apt-get install libapache-mod-php4
```

APT will automatically install the PHP 4 module for Apache 1.3, and all its dependencies and then activate it. If you're not asked to restart Apache during the install process, you'll have to do it manually :

Example #12 - Stopping and starting Apache once PHP 4 is installed

```
# /etc/init.d/apache stop
# /etc/init.d/apache start
```

Better control on configuration

In the last section, PHP was installed with only core modules. This may not be what you want and you will soon discover that you need more activated modules, like MySQL, cURL, GD, etc.

When you compile PHP from source yourself, you need to activate modules via the *configure* command. With APT, you just have to install additional packages. They're all named 'php4-*' (or 'php5-*' if you installed PHP 5 from a third party repository).

Example #13 - Getting the list of PHP additional packages

```
# dpkg -l 'php4-*'
```

As you can see from the last output, there's a lot of PHP modules that you can install (excluding the php4-cgi, php4-cli or php4-pear special packages). Look at them closely and choose what you need. If you choose a module and you do not have the proper libraries, APT will automatically install all the dependencies for you.

If you choose to add the MySQL, cURL and GD support to PHP the command will look something like this:

Example #14 - Install PHP with MySQL, cURL and GD

```
# apt-get install php4-mysql php4-curl php4-gd
```

APT will automatically add the appropriate lines to your different *php.ini* (*/etc/php4/apache/php.ini*, */etc/php4/cgi/php.ini*, etc).

Example #15 - These lines activate MySQL, cURL and GD into PHP

```
extension=mysql.so
extension=curl.so
extension=gd.so
```

You'll only have to stop/start Apache as previously to activate the modules.

Common Problems

- If you see the PHP source instead of the result the script should produce, APT has probably not included */etc/apache/conf.d/php4* in your Apache 1.3 configuration. Please ensure that the following line is present in your */etc/apache/httpd.conf* file then stop/start Apache:

  Example #16 - This line activates PHP 4 into Apache

  ```
  # Include /etc/apache/conf.d/
  ```

- If you installed an additional module and if its functions are not available in your scripts, please ensure that the appropriate line is present in your *php.ini*, as seen before. APT may fail during the installation of the additional module, due to a confusing debconf configuration.

Installation on Mac OS X

This section contains notes and hints specific to installing PHP on Mac OS X. There are two slightly different versions of Mac OS X, Client and Server, our manual deals with installing PHP on both systems. Note that PHP is not available for MacOS 9 and earlier versions.

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.

The following resources offer easy to install packages and precompiled binaries for PHP on Mac OS:

- MacPorts: » http://www.macports.org/
- Entropy: » http://www.entropy.ch/software/macosx/php/
- Fink: » http://fink.sourceforge.net/

Using the bundled PHP

PHP has come standard with Macs since OS X version 10.0.0. Enabling PHP with the default web server requires uncommenting a few lines in the Apache configuration file *httpd.conf* whereas the CGI and/or CLI are enabled by default (easily accessible via the Terminal program).

Enabling PHP using the instructions below is meant for quickly setting up a local development environment. It's *highly recommended* to always upgrade PHP to the newest version. Like most live software, newer versions are created to fix bugs and add features and PHP being is no different. See the appropriate MAC OS X installation documentation for further details. The following instructions are geared towards a beginner with details provided for getting a default setup to work. All users are encouraged to compile, or install a new packaged version.

The standard installation type is using mod_php, and enabling the bundled mod_php on Mac OS X for the Apache web server (the default web server, that is accessible via System Preferences) involves the following steps:

- Locate and open the Apache configuration file. By default, the location is as follows: */etc/httpd/httpd.conf* Using *Finder* or *Spotlight* to find this file may prove difficult as by default it's private and owned by the *root* user.

  Note

  One way to open this is by using a Unix based text editor in the Terminal, for example *nano*, and because the file is owned by root we'll use the *sudo* command to open it (as root) so for example type the following into the *Terminal* Application (after, it will prompt for a password): *sudo nano /etc/httpd/httpd.conf*

  Noteworthy nano commands: ^*w* (search), ^*o* (save), and ^*x* (exit) where ^ represents the Ctrl key.

- With a text editor, uncomment the lines (by removing the #) that look similar to the following (these two lines are often not together, locate them both in the file):
  ```
  # LoadModule php4_module libexec/httpd/libphp4.so

  # AddModule mod_php4.c
  ```
  Notice the location/path. When building PHP in the future, the above files should be replaced or commented out.

- Be sure the desired extensions will parse as PHP (examples: .php .html and .inc) Due to the following statement already existing in *httpd.conf* (as of Mac Panther), once PHP is enabled the*.php* files will automatically parse as PHP.
  ```
  <IfModule mod_php4.c>
     # If php is turned on, we respect .php and .phps files.
     AddType application/x-httpd-php .php
     AddType application/x-httpd-php-source .phps

     # Since most users will want index.php to work we
     # also automatically enable index.php
     <IfModule mod_dir.c>
         DirectoryIndex index.html index.php
     </IfModule>
  </IfModule>
  ```

- Be sure the DirectoryIndex loads the desired default index file This is also set in *httpd.conf*. Typically *index.php* and *index.html* are used. By default *index.php* is enabled because it's also in the PHP check shown above. Adjust accordingly.

- Set the *php.ini* location or use the default A typical default location on Mac OS X is */usr/local/php/php.ini* and a call to phpinfo() will reveal this information. If a *php.ini* is not used, PHP will use all default values. See also the related FAQ on finding php.ini.

- Locate or set the *DocumentRoot* This is the root directory for all the web files. Files in this directory are served from the web server so the PHP files will parse as PHP before outputting them to the browser. A typical default path is */Library/WebServer/Documents* but this can be set to anything in *httpd.conf*. Alternatively, the default DocumentRoot for individual users is */Users/yourusername/Sites*

- Create a phpinfo() file The phpinfo() function will display information about PHP. Consider creating a file in the DocumentRoot with the following PHP code:

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

- Restart Apache, and load the PHP file created above To restart, either execute *sudo apachectl graceful* in the shell or stop/start the "Personal Web Server" option in the OS X System Preferences. By default, loading local files in the browser will have an URL like so: *http://localhost/info.php* Or using the DocumentRoot in the user directory is another option and would end up looking like: *http://localhost/~yourusername/info.php*

The CLI (or CGI in older versions) is appropriately named *php* and likely exists as */usr/bin/php*. Open up the terminal, read the command line section of the PHP manual, and execute *php -v* to check the PHP version of this PHP binary. A call to phpinfo() will also reveal this information.

Compiling for OS X Server

Mac OS X Server install

- Get the latest distributions of Apache and PHP.
- 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/Header
s \
--enable-shared=max \
--enable-module=most \
--target=apache
```
- If you want the compiler to do some optimization, you may also want to add this line:
```
setenv OPTIM=-O2
```
- 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*.
- Type *make* and *make install*. This will add a directory to your Apache source directory under *src/modules/php4*.
- 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/Header
s \
--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. Run *make* and *make install* again.

- Copy and rename the *php.ini-dist* file to your *bin* directory from your PHP 4 source directory: **cp php.ini-dist /usr/local/bin/php.ini** or (if your don't have a local directory) **cp php.ini-dist /usr/bin/php.ini**.

Compiling for MacOS X Client

The following instructions will help you install a PHP module for the Apache web server included in MacOS X. This version includes support for the MySQL and PostgreSQL databases. These instructions are graciously provided by » Marc Liyanage.

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

Do this to install:

- Open a terminal window.
- Type **wget http://www.diax.ch/users/liyanage/software/macosx/libphp4.so.gz**, wait for the download to finish.
- Type **gunzip libphp4.so.gz**.
- 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.

Installation on Windows systems

This section applies to Windows 98/Me and Windows NT/2000/XP/2003. PHP will not work on 16 bit platforms such as Windows 3.1 and sometimes we refer to the supported Windows platforms as Win32. Windows 95 is no longer supported as of PHP 4.3.0.

Note
Windows 98 is no longer supported as of PHP 5.3.0.

Note
Windows 95 is no longer supported as of PHP 4.3.0.

There are two main ways to install PHP for Windows: either manually or by using the 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.

Warning
There are several all-in-one installers over the Internet, but none of those are endorsed by PHP.net, as we believe that using one of the official windows packages from » http://www.php.net/downloads.php is the best choice to have your system secure and optimized.

Windows Installer (PHP 5.2 and later)

The Windows PHP installer for later versions of PHP is built using MSI technology using the Wix Toolkit (» http://wix.sourceforge.net/). It will install and configure PHP and all the built-in and PECL extensions, as well as configure many of the popular web servers such as IIS, Apache, and Xitami.

First, install your selected HTTP (web) server on your system, and make sure that it works. Then proceed with one of the following install types.

Normal Install

Run the MSI installer and follow the instructions provided by the installation wizard. You will be prompted to select the Web Server you wish to configure first, along with any configuration details needed.

You will then be prompted to select which features and extensions you wish to install and enable. By selecting "Will be installed on local hard drive" in the drop-down menu for each item you can trigger whether to install the feature or not. By selecting "Entire feature will be installed on local hard drive", you will be able to install all sub-features of the included feature (for example by selecting this options for the feature "PDO" you will install all PDO Drivers).

Warning
It is not recommended to install all extensions by default, since many other them require dependencies from outside PHP in order to function properly. Instead, use the Installation Repair Mode that can be triggered thru the 'Add/Remove Programs' control panel to enable or disable extensions and features after installation.

The installer then sets up PHP to be used in Windows and the *php.ini* file, and configures certain web servers to use PHP. The installer will currently configure IIS, Apache, Xitami, and Sambar Server; if you are using a different web server you'll need to configure it manually.

Silent Install

The installer also supports a silent mode, which is helpful for Systems Administrators to deploy PHP easily. To use silent mode:

```
msiexec.exe /i php-VERSION-win32-install.msi /q
```

You can control the install directory by passing it as a parameter to the install. For example, to install to e:\php:

```
msiexec.exe /i php-VERSION-win32-install.msi /q INSTALLDIR=e:\php
```

You can also use the same syntax to specify the Apache Configuration Directory (APACHEDIR), the Sambar Server directory (SAMBARDIR), and the Xitami Server directory (XITAMIDIR).

You can also specify what features to install. For example, to install the mysqli extension and the CGI executable:

```
msiexec.exe /i php-VERSION-win32-install.msi /q ADDLOCAL=cgi,ext_php_mysqli
```

The current list of Features to install is as follows:

```
MainExecutable - php.exe executable
ScriptExecutable - php-win.exe executable
ext_php_* - the various extensions ( for example: ext_php_mysql for MySQL )
apache13 - Apache 1.3 module
apache20 - Apache 2.0 module
apache22 - Apache 2,2 module
apacheCGI - Apache CGI executable
iis4ISAPI - IIS ISAPI module
```

```
iis4CGI - IIS CGI executable
NSAPI - Sun/iPlanet/Netscape server module
Xitami - Xitami CGI executable
Sambar - Sambar Server ISAPI module
CGI - php-cgi.exe executable
PEAR - PEAR installer
Manual - PHP Manual in CHM Format
```

For more information on installing MSI installers from the command line, visit » http://msdn.microsoft.com/en-us/library/aa367988.aspx

Upgrading PHP with the Install

To upgrade, run the installer either graphically or from the command line as normal. The installer will read your current install options, remove your old installation, and reinstall PHP with the same options as before. It is recommended that you use this method of keeping PHP updated instead of manually replacing the files in the installation directory.

Windows Installer (PHP 5.1.0 and earlier)

The Windows PHP installer is available from the downloads page at » http://www.php.net/downloads.php. This installs the *CGI version* of PHP and for IIS, PWS, and Xitami, it configures the web server as well. The installer does not include any extra external PHP extensions (php_*.dll) as you'll only find those in the Windows Zip Package and PECL downloads.

Note
While the Windows installer is an easy way to make PHP work, it is restricted in many aspects as, for example, the automatic setup of extensions is not supported. Use of the installer isn't the preferred method for installing PHP.

First, install your selected HTTP (web) 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 certain web servers to use PHP. One of the web servers the PHP installer does not configure for is Apache, so you'll need to configure it manually.

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 with a web server on Microsoft Windows. To get started you'll need to download the zip binary distribution from the downloads page at » http://www.php.net/downloads.php.

Although there are many all-in-one installation kits, and we also distribute a PHP installer for Microsoft Windows, we recommend you take the time to setup PHP yourself as this will provide you with a better understanding of the system, and enables you to install PHP extensions easily when needed.

Note
Upgrading from a previous PHP version Previous editions of the manual suggest moving various ini and DLL files into your SYSTEM (i.e. *C:\WINDOWS*) folder and while this simplifies the installation procedure it makes upgrading difficult. We advise you remove all of these files (like *php.ini* and PHP related DLLs from the Windows SYSTEM folder) before moving on with a new PHP installation. Be sure to backup these files as you might break the entire system. The old *php.ini* might be useful in setting up the new PHP as well. And as you'll soon learn, the preferred method for installing PHP is to keep all PHP related files in one directory and have this directory available to your systems PATH.

Note
MDAC requirements If you use Microsoft *Windows 98/NT4* download the latest version of the Microsoft Data Access Components (MDAC) for your platform. MDAC is available at » http://msdn.microsoft.com/data/. This requirement exists because ODBC is built into the distributed Windows binaries.

The following steps should be completed on all installations before any server specific instructions are performed:

Extract the distribution file into a directory of your choice. If you are installing PHP 4, extract to *C:\*, as the zip file expands to a foldername like *php-4.3.7-Win32*. If you are installing PHP 5, extract to *C:\php* as the zip file doesn't expand as in PHP 4. You may choose a different location but do not have spaces in the path (like *C:\Program Files\PHP*) as some web servers will crash if you do.

The directory structure extracted from the zip is different for PHP versions 4 and 5 and look like as follows:

Example #17 - PHP 4 package structure

```
c:\php
  |
  +--cli
  |  |
  |  |-php.exe           -- CLI executable - ONLY for command line scripting
  |
  +--dlls                -- support DLLs required by some extensions
  |  |
  |  |-expat.dll
  |  |
  |  |-fdftk.dll
  |  |
  |  |-...
  |
  +--extensions          -- extension DLLs for PHP
  |  |
  |  |-php_bz2.dll
  |  |
  |  |-php_cpdf.dll
  |  |
  |  |-..
  |
  +--mibs                -- support files for SNMP
  |
  +--openssl             -- support files for Openssl
  |
  +--pdf-related         -- support files for PDF
  |
  +--sapi                -- SAPI (server module support) DLLs
  |  |
  |  |-php4apache.dll
  |  |
  |  |-php4apache2.dll
  |  |
  |  |-..
  |
  +--PEAR                -- initial copy of PEAR
  |
  |
  |-go-pear.bat          -- PEAR setup script
  |
  |-..
  |
  |-php.exe              -- CGI executable
  |
  |-..
  |
  |-php.ini-dist         -- default php.ini settings
```

```
   |
   |-php.ini-recommended  -- recommended php.ini settings
   |
   |-php4ts.dll           -- core PHP DLL
   |
   |-...
```

Or:

Example #18 - PHP 5 package structure

```
c:\php
   |
  +--dev
   |  |
   |  |-php5ts.lib
   |
  +--ext                 -- extension DLLs for PHP
   |  |
   |  |-php_bz2.dll
   |  |
   |  |-php_cpdf.dll
   |  |
   |  |-..
   |
  +--extras
   |  |
   |  +--mibs             -- support files for SNMP
   |  |
   |  +--openssl          -- support files for Openssl
   |  |
   |  +--pdf-related      -- support files for PDF
   |  |
   |  |-mime.magic
   |
  +--pear                -- initial copy of PEAR
   |
   |
   |-go-pear.bat          -- PEAR setup script
   |
   |-fdftk.dll
   |
   |-..
   |
   |-php-cgi.exe          -- CGI executable
   |
   |-php-win.exe          -- executes scripts without an opened command prompt
   |
   |-php.exe              -- CLI executable - ONLY for command line scripting
   |
   |-..
   |
   |-php.ini-dist         -- default php.ini settings
   |
   |-php.ini-recommended  -- recommended php.ini settings
   |
   |-php5activescript.dll
```

```
    |
    |-php5apache.dll
    |
    |-php5apache2.dll
    |
    |-..
    |
    |-php5ts.dll           -- core PHP DLL
    |
    |-...
```

Notice the differences and similarities. Both PHP 4 and PHP 5 have a CGI executable, a CLI executable, and server modules, but they are located in different folders and/or have different names. While PHP 4 packages have the server modules in the *sapi* folder, PHP 5 distributions have no such directory and instead they're in the PHP folder root. The supporting DLLs for the PHP 5 extensions are also not in a seperate directory.

Note

In PHP 4, you should move all files located in the *dll* and *sapi* folders to the main folder (e.g. *C:\php*).

Here is a list of server modules shipped with PHP 4 and PHP 5:

- *sapi/php4activescript.dll (php5activescript.dll)* - ActiveScript engine, allowing you to embed PHP in your Windows applications.
- *sapi/php4apache.dll (php5apache.dll)* - Apache 1.3.x module.
- *sapi/php4apache2.dll (php5apache2.dll)* - Apache 2.0.x module.
- *sapi/php5apache2_2.dll* - Apache 2.2.x module.
- *sapi/php4isapi.dll (php5isapi.dll)* - ISAPI Module for ISAPI compliant web servers like IIS 4.0/PWS 4.0 or newer.
- *sapi/php4nsapi.dll (php5nsapi.dll)* - Sun/iPlanet/Netscape server module.
- *sapi/php4pi3web.dll (no equivalent in PHP 5)* - Pi3Web server module.

Server modules provide significantly better performance and additional functionality compared to the CGI binary. The CLI version is designed to let you use PHP for command line scripting. More information about CLI is available in the chapter about using PHP from the command line.

Warning

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

The CGI and CLI binaries, and the web server modules all require the *php4ts.dll* (*php5ts.dll*) file to be available to them. You have to make sure that this file can be found by your PHP installation. The search order for this DLL is as follows:

- The same directory from where *php.exe* is called, or in case you use a SAPI module, the web server's directory (e.g. *C:\Program Files\Apache Group\Apache2\bin*).
- Any directory in your Windows *PATH* environment variable.

To make *php4ts.dll* / *php5ts.dll* available you have three options: copy the file to the Windows system directory, copy the file to the web server's directory, or add your PHP directory, *C:\php* to the *PATH*. For better maintenance, we advise you to follow the last option, add *C:\php* to the *PATH*, because it will be simpler to upgrade PHP in the future. Read more about how to add your PHP directory to *PATH* in the corresponding FAQ entry (and then don't forget to restart the computer - logoff isn't enough).

The next step is to set up a valid configuration file for PHP, *php.ini*. There are two ini files distributed in the zip file, *php.ini-dist* and *php.ini-recommended*. We advise you to use *php.ini-recommended*, because we optimized the default settings in this file for performance, and security. Read this well documented file carefully because it has changes from *php.ini-dist* that will drastically affect your setup. Some examples are display_errors being *off* and magic_quotes_gpc being *off*. In addition to reading these, study 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. Copy your chosen ini-file to a directory that PHP is able to find and rename it to *php.ini*. PHP searches for *php.ini* in the locations described in The configuration file section.

If you are running Apache 2, the simpler option is to use the PHPIniDir directive (read the installation on Apache 2 page), otherwise your best option is to set the *PHPRC* environment variable. This process is explained in the following FAQ entry.

Note

If you're using NTFS on Windows NT, 2000, XP or 2003, make sure that the user running the web server has read permissions to your *php.ini* (e.g. make it readable by Everyone).

The following steps are optional:

- Edit your new *php.ini* file. If you plan to use OmniHTTPd, do not follow the next step. Set the doc_root to point to your web servers document_root. For example:

  ```
  doc_root = c:\inetpub\wwwroot // for IIS/PWS

  doc_root = c:\apache\htdocs // for Apache
  ```

- Choose the 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 configuration setting 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. For an up-to-date *browscap.ini*, read the following FAQ.

PHP is now setup on your system. The next step is to choose a web server, and enable it to run PHP. Choose a web server from the table of contents.

ActiveScript

This section contains notes specific to the ActiveScript installation.

ActiveScript is a Windows only SAPI that enables you to use PHP script in any ActiveScript compliant host, like Windows Script Host, ASP/ASP.NET, Windows Script Components or Microsoft Scriptlet control.

As of PHP 5.0.1, ActiveScript has been moved to the » PECL repository. The DLL for this PECL extension may be downloaded from either the » PHP Downloads page or from » http://pecl4win.php.net/

Note

You should read the manual installation steps first!

After installing PHP, you should download the ActiveScript DLL (*php5activescript.dll*) and place it in the main PHP folder (e.g. *C:\php*).

After having all the files needed, you must register the DLL on your system. To achieve this, open a Command Prompt window (located in the Start Menu). Then go to your PHP directory by typing something like *cd C:\php*. To register the DLL just type *regsvr32 php5activescript.dll*.

To test if ActiveScript is working, create a new file, named *test.wsf* (the extension is very important) and type:

```
<job id="test">

<script language="PHPScript">
 $WScript->Echo("Hello World!");
</script>

</job>
```

Save and double-click on the file. If you receive a little window saying "Hello World!" you're done.

Note

In PHP 4, the engine was named 'ActivePHP', so if you are using PHP 4, you should replace 'PHPScript' with 'ActivePHP' in the above example.

Note

ActiveScript doesn't use the default *php.ini* file. Instead, it will look only in the same directory as the .exe that caused it to load. You should create *php-activescript.ini* and place it in that folder, if you wish to load extensions, etc.

Microsoft IIS / PWS

This section contains notes and hints specific to IIS (Microsoft Internet Information Server).

Warning

A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

General considerations for all installations of PHP with IIS or PWS

- First, read the Manual Installation Instructions. Do not skip this step as it provides crucial information for installing PHP on Windows.
- CGI users must set the cgi.force_redirect PHP directive to *0* inside *php.ini*. Read the faq on cgi.force_redirect for important details. Also, CGI users may want to set the cgi.redirect_status_env directive. When using directives, be sure these directives aren't commented out inside *php.ini*.
- The PHP 4 CGI is named *php.exe* while in PHP 5 it's *php-cgi.exe*. In PHP 5, *php.exe* is the CLI, and not the CGI.
- Modify the Windows *PATH* environment variable to include the PHP directory. This way the PHP DLL files and PHP executables can all remain in the PHP directory without cluttering up the Windows system directory. For more details, see the FAQ on Setting the PATH.
- The IIS user (usually IUSR_MACHINENAME) needs permission to read various files and directories, such as *php.ini*, docroot, and the session tmp directory.
- Be sure the extension_dir and doc_root PHP directives are appropriately set in *php.ini*. These directives depend on the system that PHP is being installed on. In PHP 4, the extension_dir is *extensions* while with PHP 5 it's *ext*. So, an example PHP 5

extensions_dir value is *"c:\php\ext"* and an example IIS doc_root value is *"c:\Inetpub\wwwroot"*.

- PHP extension DLL files, such as *php_mysql.dll* and *php_curl.dll*, are found in the zip package of the PHP download (not the PHP installer). In PHP 5, many extensions are part of PECL and can be downloaded in the "Collection of PECL modules" package. Files such as *php_zip.dll* and *php_ssh2.dll*. » Download PHP files here.
- When defining the executable, the 'check that file exists' box may also be checked. For a small performance penalty, the IIS (or PWS) will check that the script file exists and sort out authentication before firing up PHP. This means that the web server will provide sensible 404 style error messages instead of CGI errors complaining that PHP did not output any data.
- The PHP executable is distributed as a 32bit application. If you are running a 64bit version of Windows you will either need to rebuild the binary yourself, or make sure IIS is configured to also run 32bit extensions. You can usually turn this on by using the IIS Administration script as follows: Cscript.exe adsutil.vbs SET W3SVC/AppPools/Enable32bitAppOnWin64 1

Windows NT/200x/XP and IIS 4 or newer

PHP may be installed as a CGI binary, 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', do the following:
- Change the Execute Permissions to 'Scripts only'
- Click on the 'Configuration' button, and choose the Application Mappings tab. Click Add and set the Executable path to the appropriate CGI file. An example PHP 5 value is: *C:\php\php-cgi.exe* Supply*.php* as the extension. Leave 'Method exclusions' blank, and check the 'Script engine' checkbox. Now, click OK a few times.
- 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* / *php-cgi.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* / *php5isapi.dll*.
- Under 'Home Directory', 'Virtual Directory', or 'Directory', do the following:
- Change the Execute Permissions to 'Scripts only'

- Click on the 'Configuration' button, and choose the Application Mappings tab. Click Add and set the Executable path to the appropriate ISAPI DLL. An example PHP 5 value is: *C:\php\php5isapi.dll* Supply*.php* as the extension. Leave 'Method exclusions' blank, and check the 'Script engine' checkbox. Now, click OK a few times.
- Stop IIS completely (NET STOP iisadmin)
- Start IIS again (NET START w3svc)

With IIS 6 (2003 Server), open up the IIS Manager, go to Web Service Extensions, choose "Add a new Web service extension", enter in a name such as PHP, choose the Add button and for the value browse to either the ISAPI file (*php4isapi.dll* or *php5isapi.dll*) or CGI (*php.exe* or *php-cgi.exe*) then check "Set extension status to Allowed" and click OK.

In order to use *index.php* as a default content page, do the following: From within the Documents tab, choose Add. Type in *index.php* and click OK. Adjust the order by choosing Move Up or Move Down. This is similar to setting DirectoryIndex with Apache.

The steps above must be repeated for each extension that is to be associated with PHP scripts.*.php* is the most common although*.php3* may be required for legacy applications.

If you experience 100% CPU usage after some time, turn off the IIS setting *Cache ISAPI Application*.

Windows and PWS 4

PWS 4 does not support ISAPI, only PHP CGI should be used.

- Edit the enclosed *pws-php4cgi.reg* / *pws-php5cgi.reg* file (look into the SAPI folder for PHP 4, or in the main folder for PHP 5) to reflect the location of your *php.exe* / *php-cgi.exe*. Backslashes should be escaped, for example: *[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map] ".php"="C:\\php\\php.exe"* (change to *C:\\php\\php-cgi.exe* if you are using PHP 5) Now merge this registery file into your system; you may do this by double-clicking it.
- 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 and PWS/IIS 3

The recommended method for configuring these servers is to use the REG file included with the distribution (*pws-php4cgi.reg* in the SAPI folder for PHP 4, or *pws-php5cgi.reg* in the main folder for PHP 5). 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 "%s" %s* for PHP 4, or *C:\php\php-cgi.exe "%s" %s* for PHP 5.
- 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 from Steven Genusa to configure their script maps.

Apache 1.3.x on Microsoft Windows

This section contains notes and hints specific to Apache 1.3.x installs of PHP on Microsoft Windows systems. There are also instructions and notes for Apache 2 on a separate page.

Note
Please read the manual installation steps first!

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* for PHP 4 and *php-cgi.exe* for PHP 5), the other is to use the Apache Module DLL. In either case you need to edit your *httpd.conf* to configure Apache to work with PHP, and then restart the server.

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 Documentation for further configuration directives.

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
Remember that when adding path values in the Apache configuration files on Windows, all backslashes such as *c:\directory\file.ext* must be converted to forward slashes: *c:/directory/file.ext*. A trailing slash may also be necessary for directories.

Installing as an Apache module

You should add the following lines to your Apache *httpd.conf* file:

Example #19 - PHP as an Apache 1.3.x module

This assumes PHP is installed to *c:\php*. Adjust the path if this is not the case.

For PHP 4:

```
# Add to the end of the LoadModule section
# Don't forget to copy this file from the sapi directory!
LoadModule php4_module "C:/php/php4apache.dll"

# Add to the end of the AddModule section
AddModule mod_php4.c
```

For PHP 5:

```
# Add to the end of the LoadModule section
LoadModule php5_module "C:/php/php5apache.dll"

# Add to the end of the AddModule section
AddModule mod_php5.c
```

For both:

```
# Add this line inside the <IfModule mod_mime.c> conditional brace
AddType application/x-httpd-php .php

# For syntax highlighted .phps files, also add
AddType application/x-httpd-php-source .phps
```

Installing as a CGI binary

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:

Example #20 - PHP and Apache 1.3.x as CGI

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

# For PHP 4
Action application/x-httpd-php "/php/php.exe"

# For PHP 5
Action application/x-httpd-php "/php/php-cgi.exe"

# specify the directory where php.ini is
SetEnv PHPRC C:/php
```

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

A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

If you would like to present PHP source files syntax highlighted, there is no such convenient option as with the module version of PHP. If you chose to configure Apache to use PHP as a CGI binary, you will need to use the highlight_file() function. To do this simply create a PHP script file and add this code: *<?php highlight_file('some_php_script.php'); ?>*.

Apache 2.0.x on Microsoft Windows

This section contains notes and hints specific to Apache 2.0.x installs of PHP on Microsoft Windows systems. We also have instructions and notes for Apache 1.3.x users on a separate page.

Note

You should read the manual installation steps first!

Note

Apache 2.2.x Support

Users of Apache 2.2.x may use the documentation below except the appropriate DLL file is named *php5apache2_2.dll* and it only exists as of PHP 5.2.0. See also » http://snaps.php.net/

Warning

We do not recommend using a threaded MPM in production with Apache 2. Use the prefork MPM instead, or use Apache 1. For information on why, read the related FAQ entry on using Apache2 with a threaded MPM

You are highly encouraged to take a look at the » Apache Documentation to get a basic understanding of the Apache 2.0.x Server. Also consider to read the » Windows specific notes for Apache 2.0.x before reading on here.

Note

PHP and Apache 2.0.x compatibility notes

The following versions of PHP are known to work with the most recent version of Apache 2.0.x:

- PHP 4.3.0 or later available at » http://www.php.net/downloads.php.
- the latest stable development version. Get the source code » http://snaps.php.net/php5-latest.tar.gz or download binaries for Windows » http://snaps.php.net/win32/php5-win32-latest.zip.
- a prerelease version downloadable from » http://qa.php.net/.
- you have always the option to obtain PHP through » anonymous CVS.

These versions of PHP are compatible to Apache 2.0.40 and later.

Apache 2.0 *SAPI* -support started with PHP 4.2.0. PHP 4.2.3 works with Apache 2.0.39, don't use any other version of Apache with PHP 4.2.3. However, the recommended setup is to use PHP 4.3.0 or later with the most recent version of Apache2.

All mentioned versions of PHP will work still with Apache 1.3.x.

Warning

Apache 2.0.x is designed to run on Windows NT 4.0, Windows 2000 or Windows XP. At this time, support for Windows 9x is incomplete. Apache 2.0.x is not expected to work on those platforms at this time.

Download the most recent version of » Apache 2.0.x and a fitting PHP version. Follow the Manual Installation Steps and come back to go on with the integration of PHP and Apache.

There are two ways to set up PHP to work with Apache 2.0.x on Windows. One is to use the CGI binary the other is to use the Apache module DLL. In either case you need to edit your *httpd.conf* to configure Apache to work with PHP and then restart the server.

Note

Remember that when adding path values in the Apache configuration files on Windows, all backslashes such as *c:\directory\file.ext* must be converted to forward slashes: *c:/directory/file.ext*. A trailing slash may also be necessary for directories.

Installing as a CGI binary

You need to insert these three lines to your Apache *httpd.conf* configuration file to set up the CGI binary:

Example #21 - PHP and Apache 2.0 as CGI

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

# For PHP 4
Action application/x-httpd-php "/php/php.exe"

# For PHP 5
Action application/x-httpd-php "/php/php-cgi.exe"
```

Warning

A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

Installing as an Apache module

You need to insert these two lines to your Apache *httpd.conf* configuration file to set up the PHP module for Apache 2.0:

Example #22 - PHP and Apache 2.0 as Module

```
# For PHP 4 do something like this:
LoadModule php4_module "c:/php/php4apache2.dll"
# Don't forget to copy the php4apache2.dll file from the sapi directory!
AddType application/x-httpd-php .php

# For PHP 5 do something like this:
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php

# configure the path to php.ini
PHPIniDir "C:/php"
```

Note

Remember to substitute your actual path to PHP for the *c:/php/* in the above examples. Take care to use either *php4apache2.dll* or *php5apache2.dll* in your LoadModule directive and *not php4apache.dll* or *php5apache.dll* as the latter ones are designed to run with Apache 1.3.x.

Note
If you want to use content negotiation, read related FAQ.

Warning
Don't mix up your installation with DLL files from *different PHP versions*. You have the only choice to use the DLL's and extensions that ship with your downloaded PHP version.

Sun, iPlanet and Netscape servers on Microsoft Windows

This section contains notes and hints specific to Sun Java System Web Server, Sun ONE Web Server, iPlanet and Netscape server installs of PHP on Windows.

From PHP 4.3.3 on you can use PHP scripts with the NSAPI module to generate custom directory listings and error pages. Additional functions for Apache compatibility are also available. For support in current web servers read the note about subrequests.

CGI setup on Sun, iPlanet and Netscape servers

To install PHP as a CGI handler, 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

NSAPI setup on Sun, iPlanet and Netscape servers

To install PHP with NSAPI, 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).
- Edit *magnus.conf* (for servers >= 6) or *obj.conf* (for servers < 6) and add the following: You should place the lines after *mime types init*.
  ```
  Init fn="load-modules" funcs="php4_init,php4_execute,php4_auth_trans"
  shlib="c:/php/sapi/php4nsapi.dll"
  Init fn="php4_init" LateInit="yes" errorString="Failed to initialise PHP!"
  [php_ini="c:/path/to/php.ini"]
  ```
  (PHP >= 4.3.3) The *php_ini* parameter is optional but with it you can place your *php.ini* in your web server configuration directory.
- Configure the default object in *obj.conf* (for virtual server classes [Sun Web Server 6.0+] in their *vserver.obj.conf*): 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" [inikey=value
  inikey=value ...]
  ```
  (PHP >= 4.3.3) As additional parameters you can add some special *php.ini* -values, for example you can set a *docroot="/path/to/docroot"* specific to the context *php4_execute* is called. For boolean ini-keys please use 0/1 as value, not *"On","Off",...* (this will not work correctly), e.g. *zlib.output_compression=1* instead of *zlib.output_compression="On"*
- This is only needed if you want to configure a directory that only consists of PHP scripts (same like a cgi-bin directory):
  ```
  <Object name="x-httpd-php">
  ObjectType fn="force-type" type="magnus-internal/x-httpd-php"
  Service fn=php4_execute [inikey=value inikey=value ...]
  </Object>
  ```
  After that you can configure a directory in the Administration server and assign it the style *x-httpd-php*. All files in it will get executed as PHP. This is nice to hide PHP usage by renaming files to*.html*.
- Restart your web service and apply changes
- Do it for each web server instance you want PHP to run

Note
More details about setting up PHP as an NSAPI filter can be found here: [» http://benoit.noss.free.fr/php/install-php4.html](http://benoit.noss.free.fr/php/install-php4.html)

Note
The stacksize that PHP uses depends on the configuration of the web server. If you get crashes with very large PHP scripts, it is recommended to raise it with the Admin Server (in the section "MAGNUS EDITOR").

CGI environment and recommended modifications in php.ini

Important when writing PHP scripts is the fact that Sun JSWS/Sun ONE WS/iPlanet/Netscape is a multithreaded web server. Because of that all requests are running in the same process space (the space of the web server itself) and this space has only one environment. If you want to get CGI variables like *PATH_INFO*, *HTTP_HOST* etc. it is not the correct way to try this in the old PHP way with getenv() or a similar way (register globals to environment, *$_ENV*). You would only get the environment of the running web server without any valid CGI variables!

Note
Why are there (invalid) CGI variables in the environment? Answer: This is because you started the web server process from the admin server which runs the startup script of the web server, you wanted to start, as a CGI script (a CGI script inside of the admin server!). This is why the environment of the started web server has some CGI environment variables in it. You can test this by starting the web server not from the administration server. Use the command line as root user and start it manually - you will see there are no CGI-like environment variables.

Simply change your scripts to get CGI variables in the correct way for PHP 4.x by using the superglobal *$_SERVER*. If you have older scripts which use *$HTTP_HOST*, etc., you should turn on *register_globals* in *php.ini* and change the variable order too (important: remove *"E"* from it, because you do not need the environment here):

```
variables_order = "GPCS"
register_globals = On
```

Special use for error pages or self-made directory listings (PHP >= 4.3.3)

You can use PHP to generate the error pages for *"404 Not Found"* or similar. Add the following line to the object in *obj.conf* for every error page you want to overwrite:

```
Error fn="php4_execute" code=XXX script="/path/to/script.php" [inikey=value
inikey=value...]
```

where *XXX* is the HTTP error code. Please delete any other *Error* directives which could interfere with yours. If you want to place a page for all errors that could exist, leave the *code* parameter out. Your script can get the HTTP status code with *$_SERVER['ERROR_TYPE']*.

Another possibility is to generate self-made directory listings. Just create a PHP script which displays a directory listing and replace the corresponding default Service line for *type="magnus-internal/directory"* in *obj.conf* with the following:

```
Service fn="php4_execute" type="magnus-internal/directory"
script="/path/to/script.php" [inikey=value inikey=value...]
```

For both error and directory listing pages the original URI and translated URI are in the variables *$_SERVER['PATH_INFO']* and *$_SERVER['PATH_TRANSLATED']*.

Note about nsapi_virtual() and subrequests (PHP >= 4.3.3)

The NSAPI module now supports the nsapi_virtual() function (alias: virtual()) to make subrequests on the web server and insert the result in the web page. The problem is, that this function uses some undocumented features from the NSAPI library.

Under Unix this is not a problem, because the module automatically looks for the needed functions and uses them if available. If not, nsapi_virtual() is disabled.

Under Windows limitations in the DLL handling need the use of a automatic detection of the most recent *ns-httpdXX.dll* file. This is tested for servers till version 6.1. If a newer version of the Sun server is used, the detection fails and nsapi_virtual() is disabled.

If this is the case, try the following: Add the following parameter to *php4_init* in *magnus.conf* / *obj.conf*:
```
Init fn=php4_init ... server_lib="ns-httpdXX.dll"
```
where *XX* is the correct DLL version number. To get it, look in the server-root for the correct DLL name. The DLL with the biggest filesize is the right one.

You can check the status by using the phpinfo() function.

Note

But be warned: Support for nsapi_virtual() is EXPERIMENTAL!!!

OmniHTTPd Server

This section contains notes and hints specific to » OmniHTTPd on Windows.

Note

You should read the manual installation steps first!

Warning

A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

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.

Note

Important for CGI users

Read the faq on cgi.force_redirect for important details. This directive needs to be set to *0*.

- Install OmniHTTPd server.
- Right click on the blue OmniHTTPd icon in the system tray and select *Properties*
- Click on *Web Server Global Settings*
- On the 'External' tab, enter: *virtual = .php* | *actual = c:\php\php.exe* (use *php-cgi.exe* if installing PHP 5), and use the Add button.
- On the *Mime* tab, enter: *virtual = wwwserver/stdcgi* | *actual = .php*, and use the Add button.
- 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.

Sambar Server on Microsoft Windows

This section contains notes and hints specific to the » Sambar Server for Windows.

Note

You should read the manual installation steps first!

This list describes how to set up the ISAPI module to work with the Sambar server on Windows.

- Find the file called *mappings.ini* (in the config directory) in the Sambar install directory.
- Open *mappings.ini* and add the following line under *[ISAPI]*:

Example #23 - ISAPI configuration of Sambar

```
#for PHP 4
*.php = c:\php\php4isapi.dll

#for PHP 5
*.php = c:\php\php5isapi.dll
```

(This line assumes that PHP was installed in *c:\php*.)

- Now restart the Sambar server for the changes to take effect.

Note

If you intend to use PHP to communicate with resources which are held on a different computer on your network, then you will need to alter the account used by the Sambar Server Service. The default account used for the Sambar Server Service is LocalSystem which will not have access to remote resources. The account can be amended by using the Services option from within the Windows Control Panel Administation Tools.

Xitami on Microsoft Windows

This section contains notes and hints specific to » Xitami on Windows.

Note

You should read the manual installation steps first!

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

Note

Important for CGI users

Read the faq on cgi.force_redirect for important details. This directive needs to be set to *0*. If you want to use *$_SERVER['PHP_SELF']* you have to enable the cgi.fix_pathinfo directive.

Warning

A server deployed in CGI mode is open to several possible vulnerabilities. Please read our CGI security section to learn how to defend yourself from such attacks.

- Make sure the web server 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 CGI executable i.e. *C:\php\php.exe* for PHP 4, or *C:\php\php-cgi.exe* for PHP 5.
- Press the 'Save' icon.
- Restart the server to reflect changes.

Building from source

This chapter teaches how to compile PHP from sources on windows, using Microsoft's tools. To compile PHP with cygwin, please refer to Installation on Unix systems.

Quick Guide to Building On Windows

This step-by-step quick-start guide was written in March of 2008, running Windows XP Service Pack 2 with all the latest updates and building PHP 5.2.5 and PHP 5.3. Experiences using different tools may differ.

- Download and install:
  - » Microsoft Visual C++ 2008 Express Edition
  - » Windows SDK for Windows Server 2008 and .NET Framework 3.5

- Copy *C:\Program Files\Microsoft SDKs\Windows\v6.1\Include\WinResrc.h* to *C:\Program Files\Microsoft SDKs\Windows\v6.1\Include\winres.h*.
- Create the directory *C:\work*.
- Download » the Windows build tools and unzip the contents into *C:\work*.
- Create the directory *C:\usr\local\lib*. Copy the *C:\work\win32build\bin\bison.simple* into the new directory.
- Download » the Windows DNS resolver library and unzip the contents into *C:\work*.
- Open *C:\work\bindlib_w32\bindlib.dsw*. If and when asked whether to update the project, choose Yes. Choose either Debug or Release configuration in the top toolbar, then choose Build => Build Solution.
- Obtain a copy of the PHP source and extract it into the *C:\work* directory. At this point, that directory should look something like this:

```
+-C:\work
| +-bindlib_w32
| | +-Debug
| | | +-resolv.lib
| | | +-...
| | +-...
| +-win32build
| | +-bin
| | +-include
| | +-lib
| +-php-5.2.5
| | +-build
| | +-win32
| | +-...
```

- Open a shell using the Visual Studio 2008 Command Prompt shortcut in the Start menu.
- Execute the command:
```
C:\Program Files\Microsoft Visual Studio 9.0\VC> set
"PATH=C:\work\win32build\bin;%PATH%"
C:\Program Files\Microsoft Visual Studio 9.0\VC>
```
- Enter the *C:\work\php-5.2.5* directory.
- Run *cscript /nologo win32\build\buildconf.js*.
- Run *cscript /nologo configure.js --disable-all --enable-cli --enable-cgi --enable-object-out-dir=.. --disable-ipv6*. To enable debugging, add *--enable-debug* to the end.
- Run *nmake*.
- If all went well, there will now be a working PHP CLI executable at *C:\work\Debug_TS\php.exe*, or *C:\work\Release_TS\php.exe*.

Build Environment

To compile and build PHP you need a Microsoft Development Environment. The following environments are supported:

- Microsoft Visual C++ 6.0 (official)
- Microsoft Visual C++ .NET
- Microsoft Visual C++ 2005, Windows Platform SDK and .NET Framework SDK (current)

While VC6 (Microsoft Visual C++ 6.0) is used to perform official Windows builds, it can no longer be downloaded from Microsoft's website. New users seeking to build PHP for free must use Microsoft Visual C++ 2005 Express Edition and its auxiliary components.

Setting up Microsoft Visual C++ 2005 Express

Note

Combined, these components are very large and will require over one gigabyte of disk space.

Setting up Microsoft Visual C++ 2005 Express is rather involved, and requires the installation of three separate packages and various compatibility changes. Be sure to keep track of the paths in which these programs are installed into. Download and install the following programs:

- » Microsoft Visual C++ 2005 Express
- » Microsoft Windows Server 2005 Platform SDK
- » .NET Framework 2.0 Software Development Kit

There are a few post-installation steps:

- » MSVC 2005 Express must be configured to use Windows Platform SDK. It is not necessary to perform step two, as PHP does not use the graphical user interface.
- Windows Platform SDK contains a file named *WinResrc.h* usually in *Include* folder inside the SDK's installation directory. This needs to be copied and renamed to *winres.h*, the name PHP uses for the file.

Finally, when using MSVC 2005 Express from the command line, several environment variables must be set up. *vsvars32.bat* usually found in *C:\Program Files\Microsoft Visual Studio 8\Common7\Tools* (search for the file if otherwise) contains these declarations. The *PATH*, *INCLUDE* and *LIB* environment variables need the corresponding *bin*, *include* and *lib* directories of the two newly installed SDKs added to them.

Note

The .NET SDK paths may already be present in the *vsvars32.bat* file, as this SDK installs itself into the same directory as Microsoft Visual C++ 2005 Express.

Libraries

To extract the downloaded files you will also need a ZIP extraction utility. Windows XP and newer already include this functionality built-in.

Before you get started, you have to download:

- the win32 buildtools from the PHP site at » http://www.php.net/extra/win32build.zip.

- the source code for the DNS name resolver used by PHP from » http://www.php.net/extra/bindlib_w32.zip. This is a replacement for the *resolv.lib* library included in *win32build.zip*.
- If you plan to compile PHP as a Apache module you will also need the » Apache sources.

Finally, you are going to need the source to PHP itself. You can get the latest development version using » anonymous CVS, a » snapshot or the most recent released » source tarball.

Putting it all together

After downloading the required packages you have to extract them in a proper place:

- Create a working directory where all files end up after extracting, e.g: *C:\work*.
- Create the directory *win32build* under your working directory (*C:\work*) and unzip *win32build.zip* into it.
- Create the directory *bindlib_w32* under your working directory (*C:\work*) and unzip *bindlib_w32.zip* into it.
- Extract the downloaded PHP source code into your working directory (*C:\work*).
- Build the libraries you are going to need (or download the binaries if available) and place the headers and libs in the *C:\work\win32build\include* and *C:\work\win32build\lib* directories, respectively.
- If you don't have cygwin installed with bison and flex, you also need to make the *C:\work\win32build\bin* directory available in the PATH, so that thoses tools can be found by the configure script.

Following this steps your directory structure looks like this:

```
+--C:\work
|  |
|  +--bindlib_w32
|  |  |
|  |  +--arpa
|  |  |
|  |  +--conf
|  |  |
|  |  +--...
|  |
|  +--php-5.x.x
|  |  |
|  |  +--build
|  |  |
|  |  +--...
|  |  |
|  |  +--win32
|  |  |
|  |  +--...
|  |
|  +--win32build
|  |  |
|  |  +--bin
|   |   |
|   |   +--include
|   |   |
|   |   +--lib
```

If you aren't using [» Cygwin](), you must also create the directories *C:\usr\local\lib* and then copy *bison.simple* from *C:\work\win32build\bin* to *C:\usr\local\lib*.

> **Note**
>
> If you want to use PEAR and the comfortable command line installer, the CLI-SAPI is mandatory. For more information about PEAR and the installer read the documentation at the [» PEAR]() website.

Build resolv.lib

You must build the *resolv.lib* library. Decide whether you want to have debug symbols available (bindlib - Win32 Debug) or not (bindlib - Win32 Release), but please remember the choice you made, because the debug build will only link with PHP when it is also built in debug mode. Build the appropriate configuration:

- For GUI users, launch VC++, by double-clicking in *C:\work\bindlib_w32\bindlib.dsw*. Then 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 commands:
  - *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 *C:\work\bindlib_w32\Debug* or *Release* subdirectories. Copy this file into your *C:\work\win32build\lib* directory over the file by the same name found in there.

Building PHP using the new build system [PHP >=5 only]

This chapter explains how to compile PHP >=5 using the new build system, which is CLI-based and very similar with the main PHP's Unix build system.

> **Note**
>
> This build system isn't available in PHP 4. Please refer to [Building PHP using DSW files [PHP 4]]() instead.

Before starting, be sure you have read Putting it all together and you have built all needed libraries, like » Libxml or » ICU (needed for PHP >= 6).

First you should open a Visual Studio Command Prompt, which should be available under the Start menu. A regular Command Prompt window shouldn't work, as probably it doesn't have the necessary environment variables set. Then type something like *cd C:\work\php-5.x.x* to enter in the PHP source dir. Now you are ready to start configuring PHP.

The second step is running the *buildconf* batch file to make the configure script, by scanning the folder for *config.w32* files. By default this command will also search in the following directories: *pecl; ..\pecl; pecl\rpc; ..\pecl\rpc*. Since PHP 5.1.0, you can change this behaviour by using the *--add-modules-dir* argument (e.g. *cscript /nologo win32/build/buildconf.js --add-modules-dir=../php-gtk2 --add-modules-dir=../pecl*).

The third step is configuring. To view the list of the available configuration options type *cscript /nologo configure.js --help*. After choosing the options that you will enable/disable, type something like: *cscript /nologo configure.js --disable-foo --enable-fun-ext*. Using *--enable-foo=shared* will attempt to build the 'foo' extension as a shared, dynamically loadable module.

The last step is compiling. To achieve this just issue the command *nmake*. The generated files (e.g. .exe and .dll) will be placed in either *Release_TS* or *Debug_TS* directories (if built with Thread safety), or in the *Release* or *Debug* directories otherwise.

Optionally you may also run PHP's test suite, by typing *nmake test*. If you want to run just a specific test, you may use the 'TESTS' variable (e.g. *nmake /D TESTS=ext/sqlite/tests test* - will only run sqlite's tests). To delete the files that were created during the compilation, you can use the *nmake clean* command.

A very useful configure option to build snapshots is *--enable-snapshot-build*, which add a new compiling mode (*nmake build-snap*). This tries to build every extension available (as shared, by default), but it will ignore build errors in individual extensions or SAPI.

Building PHP using DSW files [PHP 4]

Compiling PHP using the DSW files isn't supported as of PHP 5, as a much more flexible system was made available. Anyway, you can still use them, but keep in mind that they are not maintained very often, so you can have compiling problems. To compile PHP 4 for windows, this is the only available way though.

Configure MVC ++

The first step is to configure MVC++ to prepare for compiling. 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. Your entries should look like this:

- Executable files: *C:\work\win32build\bin*, Cygwin users: *C:\cygwin\bin*

- Include files: *C:\work\win32build\include*
- Library files: *C:\work\win32build\lib*

Compiling

The best way to get started is to build the CGI version:

- For GUI users, launch VC++, and then select File => Open Workspace and select *C:\work\php-4.x.x\win32\php4ts.dsw*. Then select Build=>Set Active Configuration and select the desired configuration, either *php4ts - Win32 Debug_TS* or *php4ts - Win32 Release_TS*. 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 commands from the *C:\work\php-4.x.x\win32* directory:
  - *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 *C:\work\php-4.x.x\Debug_TS* or *Release_TS* subdirectories.

It is possible to do minor customization to the build process by editing the *main/config.win32.h* file. For example you can change the default location of *php.ini*, the built-in extensions, and the default location for your extensions.

Next you may want to build the CLI version which is designed to use PHP from the command line. The steps are the same as for building the CGI version, except you have to select the *php4ts_cli - Win32 Debug_TS* or *php4ts_cli - Win32 Release_TS* project file. After a successful compiling run you will find the *php.exe* in either the directory *Release_TS\cli\* or *Debug_TS\cli\*.

In order to build the SAPI module (*php4isapi.dll*) for integrating PHP with Microsoft IIS, set your active configuration to *php4isapi-whatever-config* and build the desired dll.

Installation of extensions on Windows

After installing PHP and a web server on Windows, you will probably want to install some extensions for added functionality. You can choose which extensions you would like to load when PHP starts by modifying your *php.ini*. You can also load a module dynamically in your script using dl().

The DLLs for PHP extensions are prefixed with *php_*.

Many extensions are *built into* the Windows version of PHP. This means additional DLL files, and the extension directive, are *not* used to load these extensions. The Windows PHP Extensions table lists extensions that require, or used to require, additional PHP DLL files. Here's a list of built in extensions:

In PHP 4 (updated PHP 4.3.11): BCMath, Caledar, COM, Ctype, FTP, MySQL, ODBC, Overload, PCRE, Session, Tokenizer, WDDX, XML and Zlib

In PHP 5 (updated PHP 5.0.4), the following changes exist. Built in: DOM, LibXML, Iconv, SimpleXML, SPL and SQLite. And the following are no longer built in: MySQL and Overload.

The default location PHP searches for extensions is *C:\php4\extensions* in PHP 4 and *C:\php5* in PHP 5. To change this setting to reflect your setup of PHP edit your *php.ini* file:

- You will need to change the extension_dir setting to point to the directory where your extensions lives, or where you have placed your *php_*.dll* files. For example:

  ```
  extension_dir = C:\php\extensions
  ```

- Enable the extension(s) in *php.ini* you want to use by uncommenting the *extension=php_*.dll* lines in *php.ini*. This is done by deleting the leading ; from the extension you want to load.

  Example #24 - Enable Bzip2 extension for PHP-Windows

  ```
  // change the following line from ...
  ;extension=php_bz2.dll

  // ... to
  extension=php_bz2.dll
  ```

- Some of the extensions need extra DLLs to work. Couple of them can be found in the distribution package, in the *C:\php\dlls\* folder in PHP 4 or in the main folder in PHP 5, but some, for example Oracle (*php_oci8.dll*) require DLLs which are not bundled with the distribution package. If you are installing PHP 4, copy the bundled DLLs from *C:\php\dlls* folder to the main *C:\php* folder. Don't forget to include *C:\php* in the system *PATH* (this process is explained in a separate FAQ entry).
- Some of these DLLs are not bundled with the PHP distribution. See each extensions documentation page for details. Also, read the manual section titled Installation of PECL extensions for details on PECL. An increasingly large number of PHP extensions are found in PECL, and these extensions require a separate download.

Note

If you are running a server module version of PHP remember to restart your web server to reflect your changes to *php.ini*.

The following table describes some of the extensions available and required additional dlls.

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_crack.dll	Crack functions	None
php_ctype.dll	ctype family functions	Built in since PHP 4.3.0
php_curl.dll	CURL, Client URL library functions	Requires: *libeay32.dll*, *ssleay32.dll* (bundled)
php_dba.dll	DBA: DataBase (dbm-style) Abstraction layer functions	None
php_dbase.dll	dBase functions	None
php_dbx.dll	dbx functions	
php_domxml.dll	DOM XML functions	PHP <= 4.2.0 requires: *libxml2.dll* (bundled) PHP >= 4.3.0 requires: *iconv.dll* (bundled)
php_dotnet.dll	.NET functions	PHP <= 4.1.1
php_exif.dll	EXIF functions	php_mbstring.dll. And, *php_exif.dll* must be loaded *after php_mbstring.dll* in *php.ini*.
php_fbsql.dll	FrontBase functions	PHP <= 4.2.0
php_fdf.dll	FDF: Forms Data Format functions.	Requires: *fdftk.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	Removed in PHP 4.3.2. Also note that truecolor functions are not available in GD1, instead, use *php_gd2.dll*.
php_gd2.dll	GD library image functions	GD2

php_gettext.dll	Gettext functions	PHP <= 4.2.0 requires *gnu_gettext.dll* (bundled), PHP >= 4.2.3 requires *libintl-1.dll*, *iconv.dll* (bundled).
php_hyperwave.dll	HyperWave functions	None
php_iconv.dll	ICONV characterset conversion	Requires: *iconv-1.3.dll* (bundled), PHP >=4.2.1 *iconv.dll*
php_ifx.dll	Informix functions	Requires: Informix libraries
php_iisfunc.dll	IIS management functions	None
php_imap.dll	IMAP POP3 and NNTP functions	None
php_ingres.dll	Ingres II functions	Requires: Ingres II libraries
php_interbase.dll	InterBase functions	Requires: *gds32.dll* (bundled)
php_java.dll	Java functions	PHP <= 4.0.6 requires: *jvm.dll* (bundled)
php_ldap.dll	LDAP functions	PHP <= 4.2.0 requires *libsasl.dll* (bundled), PHP >= 4.3.0 requires *libeay32.dll*, *ssleay32.dll* (bundled)
php_mbstring.dll	Multi-Byte String functions	None
php_mcrypt.dll	Mcrypt Encryption functions	Requires: *libmcrypt.dll*
php_mhash.dll	Mhash functions	PHP >= 4.3.0 requires: *libmhash.dll* (bundled)
php_mime_magic.dll	Mimetype functions	Requires: *magic.mime* (bundled)
php_ming.dll	Ming functions for Flash	None
php_msql.dll	mSQL functions	Requires: *msql.dll* (bundled)
php_mssql.dll	MSSQL functions	Requires: *ntwdblib.dll* (bundled)
php_mysql.dll	MySQL functions	PHP >= 5.0.0, requires *libmysql.dll* (bundled)

php_mysqli.dll	MySQLi functions	PHP >= 5.0.0, requires *libmysql.dll* (*libmysqli.dll* in PHP <= 5.0.2) (bundled)
php_oci8.dll	Oracle 8 functions	Requires: Oracle 8.1+ client libraries
php_openssl.dll	OpenSSL functions	Requires: *libeay32.dll* (bundled)
php_overload.dll	Object overloading functions	Built in since PHP 4.3.0
php_pdf.dll	PDF functions	None
php_pgsql.dll	PostgreSQL functions	None
php_printer.dll	Printer functions	None
php_shmop.dll	Shared Memory functions	None
php_snmp.dll	SNMP get and walk functions	NT only!
php_soap.dll	SOAP functions	PHP >= 5.0.0
php_sockets.dll	Socket functions	None
php_sybase_ct.dll	Sybase functions	Requires: Sybase client libraries
php_tidy.dll	Tidy functions	PHP >= 5.0.0
php_tokenizer.dll	Tokenizer functions	Built in since PHP 4.3.0
php_w32api.dll	W32api functions	None
php_xmlrpc.dll	XML-RPC functions	PHP >= 4.2.1 requires: *iconv.dll* (bundled)
php_xslt.dll	XSLT functions	PHP <= 4.2.0 requires *sablot.dll*, *expat.dll* (bundled). PHP >= 4.2.1 requires *sablot.dll*, *expat.dll*, *iconv.dll* (bundled).
php_yaz.dll	YAZ functions	Requires: *yaz.dll* (bundled)
php_zip.dll	Zip File functions	Read only access
php_zlib.dll	ZLib compression functions	Built in since PHP 4.3.0

Installation of PECL extensions

Introduction to PECL Installations

» PECL is a repository of PHP extensions that are made available to you via the » PEAR packaging system. This section of the manual is intended to demonstrate how to obtain and install PECL extensions.

These instructions assume */your/phpsrcdir/* is the path to the PHP source distribution, and that *extname* is the name of the PECL extension. Adjust accordingly. These instructions also assume a familiarity with the » pear command. The information in the PEAR manual for the *pear* command also applies to the *pecl* command.

To be useful, a shared extension must be built, installed, and loaded. The methods described below provide you with various instructions on how to build and install the extensions, but they do not automatically load them. Extensions can be loaded by adding an extension directive. To this *php.ini* file, or through the use of the dl() function.

When building PHP modules, it's important to have known-good versions of the required tools (autoconf, automake, libtool, etc.) See the » Anonymous CVS Instructions for details on the required tools, and required versions.

Downloading PECL extensions

There are several options for downloading PECL extensions, such as:

- » http://pecl.php.net/ The PECL web site contains information about the different extensions that are offered by the PHP Development Team. The information available here includes: ChangeLog, release notes, requirements and other similar details.
- *pecl download extname* PECL extensions that have releases listed on the PECL web site are available for download and installation using the » pecl command. Specific revisions may also be specified.
- CVS Most PECL extensions also reside in CVS. A web-based view may be seen at » http://cvs.php.net/pecl/. To download straight from CVS, the following sequence of commands may be used. Note that *phpfi* is the password for user *cvsread*:
```
$ cvs -d:pserver:cvsread@cvs.php.net:/repository login $ cvs
-d:pserver:cvsread@cvs.php.net:/repository co pecl/extname
```
- Windows downloads Windows users may find compiled PECL binaries by downloading the *Collection of PECL modules* from the » PHP Downloads page, or by retrieving a » PECL Snapshot or an extension DLL on » PECL4WIN. To compile PHP under Windows, read the appropriate chapter.

PECL for Windows users

As with any other PHP extension DLL, installation is as simple as copying the PECL extension DLLs into the extension_dir folder and loading them from *php.ini*. For example, add the following line to your *php.ini*:

```
extension=php_extname.dll
```

After doing this, restart the web service.

Compiling shared PECL extensions with the pecl command

PECL makes it easy to create shared PHP extensions. Using the » pecl command, do the following:

```
$ pecl install extname
```

This will download the source for *extname*, compile, and install *extname.so* into your extension_dir. *extname.so* may then be loaded via *php.ini*

By default, the *pecl* command will not install packages that are marked with the *alpha* or *beta* state. If no *stable* packages are available, you may install a *beta* package using the following command:

```
$ pecl install extname-beta
```

You may also install a specific version using this variant:

```
$ pecl install extname-0.1
```

Note
After enabling the extension in *php.ini*, restarting the web service is required for the changes to be picked up.

Compiling shared PECL extensions with phpize

Sometimes, using the *pecl* installer is not an option. This could be because you're behind a firewall, or it could be because the extension you want to install is not available as a PECL compatible package, such as unreleased extensions from CVS. If you need to build such an extension, you can use the lower-level build tools to perform the build manually.

The *phpize* command is used to prepare the build environment for a PHP extension. In the following sample, the sources for an extension are in a directory named *extname*:

```
$ cd extname
$ phpize
$ ./configure
$ make
# make install
```

A successful install will have created *extname.so* and put it into the PHP extensions directory. You'll need to and adjust *php.ini* and add an *extension=extname.so* line before you can use the extension.

If the system is missing the *phpize* command, and precompiled packages (like RPM's) are used, be sure to also install the appropriate *devel* version of the PHP package as they often include the *phpize* command along with the appropriate header files to build PHP and its extensions.

Execute *phpize --help* to display additional usage information.

Compiling PECL extensions statically into PHP

You might find that you need to build a PECL extension statically into your PHP binary. To do this, you'll need to place the extension source under the *php-src/ext/* directory and tell the PHP build system to regenerate its configure script.

```
$ cd /your/phpsrcdir/ext
$ pecl download extname
$ gzip -d < extname.tgz | tar -xvf -
$ mv extname-x.x.x extname
```

This will result in the following directory:

```
/your/phpsrcdir/ext/extname
```

From here, force PHP to rebuild the configure script, and then build PHP as normal:

```
$ cd /your/phpsrcdir $ rm configure $ ./buildconf --force $ ./configure --help $
./configure --with-extname --enable-someotherext --with-foobar $ make $ make
install
```

Note

To run the 'buildconf' script you need autoconf 2.13 and automake 1.4+ (newer versions of autoconf may work, but are not supported).

Whether *--enable-extname* or *--with-extname* is used depends on the extension. Typically an extension that does not require external libraries uses *--enable*. To be sure, run the following after buildconf:

```
$ ./configure --help | grep extname
```

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/support.php. To subscribe to the PHP installation mailing list, send an empty mail to » 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, safe mode, 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 document before submitting any bug reports!

Runtime Configuration

The configuration file

The configuration file (*php.ini*) 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 and CLI version, it happens on every invocation.

php.ini is searched in these locations (in order):

- SAPI module specific location (*PHPIniDir* directive in Apache 2, *-c* command line option in CGI and CLI, *php_ini* parameter in NSAPI, *PHP_INI_PATH* environment variable in THTTPD)
- The *PHPRC* environment variable. Before PHP 5.2.0 this was checked after the registry key mentioned below.
- As of PHP 5.2.0, the following registry locations are searched in order: *HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y.z\IniFilePath*, *HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y\IniFilePath* and *HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x\IniFilePath*, where x, y and z mean the PHP major, minor and release versions.
- *HKEY_LOCAL_MACHINE\SOFTWARE\PHP\IniFilePath* (Windows Registry location)
- Current working directory (except CLI)
- The web server's directory (for SAPI modules), or directory of PHP (otherwise in Windows)
- Windows directory (*C:\windows* or *C:\winnt*) (for Windows), or *--with-config-file-path* compile time option

If *php-SAPI.ini* exists (where SAPI is used SAPI, so the filename is e.g. *php-cli.ini* or *php-apache.ini*), it's used instead of *php.ini*. SAPI name can be determined by php_sapi_name().

Note

The Apache web server changes the directory to root at startup causing PHP to attempt to read *php.ini* from the root filesystem if it exists.

The *php.ini* directives handled by extensions are documented respectively on the pages of the extensions themselves. The list of the core directives is available in the appendix. Probably not all PHP directives are documented in the manual though. For a complete list of directives available in your PHP version, please read your well commented *php.ini* file. Alternatively, you may find the » the latest *php.ini* from CVS helpful too.

Example #25 - *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
track_errors = 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"
```

Since PHP 5.1.0, it is possible to refer to existing .ini variables from within .ini files. Example: *open_basedir = ${open_basedir} ":/new/dir"*.

How to change configuration settings

Running PHP as an Apache module

When using PHP as an Apache module, you can also change the configuration settings using directives in Apache configuration files (e.g. *httpd.conf*) and*.htaccess* files. You will need "AllowOverride Options" or "AllowOverride All" privileges to do so.

There are several Apache directives that allow you to change the PHP configuration from within the Apache configuration files. For a listing of which directives are **PHP_INI_ALL**, **PHP_INI_PERDIR**, or **PHP_INI_SYSTEM**, have a look at the List of php.ini directives appendix.

<variablelist><varlistentry><term> **php_value** name value</term>

- Sets the value of the specified directive. Can be used only with **PHP_INI_ALL** and **PHP_INI_PERDIR** type directives. To clear a previously set value use *none* as the value.

Note

Don't use **php_value** to set boolean values. **php_flag** (see below) should be used instead.

</varlistentry><varlistentry><term> **php_flag** name on|off</term>

- Used to set a boolean configuration directive. Can be used only with **PHP_INI_ALL** and **PHP_INI_PERDIR** type directives.

</varlistentry><varlistentry><term> **php_admin_value** name value</term>

- Sets the value of the specified directive. This *can not be used* in*.htaccess* files. Any directive type set with **php_admin_value** can not be overridden by*.htaccess*. To clear a previously set value use *none* as the value.

</varlistentry><varlistentry><term> **php_admin_flag** name on|off</term>

- Used to set a boolean configuration directive. This *can not be used* in*.htaccess* files. Any directive type set with **php_admin_flag** can not be overridden by*.htaccess*.

</varlistentry></variablelist>

Example #26 - Apache configuration example

```
<IfModule mod_php5.c>
 php_value include_path ".:/usr/local/lib/php"
 php_admin_flag safe_mode on
</IfModule>
<IfModule mod_php4.c>
 php_value include_path ".:/usr/local/lib/php"
 php_admin_flag safe_mode on
</IfModule>
```

Caution

PHP constants do not exist outside of PHP. For example, in *httpd.conf* you can not use PHP constants such as **E_ALL** or **E_NOTICE** to set the error_reporting directive as they will have no meaning and will evaluate to *0*. Use the associated bitmask values instead. These constants can be used in *php.ini*

Changing PHP configuration via the Windows registry

When running PHP on Windows, the configuration values can be modified on a per-directory basis using the Windows registry. The configuration values are stored in the registry key *HKLM\SOFTWARE\PHP\Per Directory Values*, in the sub-keys corresponding to the path names. For example, configuration values for the directory *c:\inetpub\wwwroot* would be stored in the key *HKLM\SOFTWARE\PHP\Per Directory Values\c\inetpub\wwwroot*. The settings for the directory would be active for any script running from this directory or any subdirectory of it. The values under the key should have the name of the PHP configuration directive and the string value. PHP constants in the values are not parsed. However, only configuration values changeable in **PHP_INI_USER** can be set this way, **PHP_INI_PERDIR** values can not.

Other interfaces to PHP

Regardless of how you run PHP, you can change certain values at runtime of your scripts through ini_set(). See the documentation on the ini_set() page for more information.

If you are interested in a complete list of configuration settings on your system with their current values, you can execute the phpinfo() function, and review the resulting page. You can also access the values of individual configuration directives at runtime using ini_get() or get_cfg_var().