Building Apache

Release 5.00

Afra Ahmad

April 12, 2004

E-mail: afra at prongs.org

Abstract

This document covers how to install the popular web server, Apache with other modules (mod_perl, mod_python, mod_php) under Sun Solaris, FreeBSD and Linux. In addition, explanation of how to install mod_ssl, mm and OpenSSL is covered, all of which are required for SSL integration with Apache.

The Apache 1.3.x series is only discussed in this document, as Apache 2.0, and the add-on modules, are currently still in the beta developmental phase.

The latest version of this document can be read at *http://www.prongs.org/linux/apache*.

The PDF version of this very document is available at *http://www.prongs.org/linux/apache/apache.pdf*. Please email me if you have any comments of questions.

Contents

1 Introduction

This document should help you, the system administrator, compile Apache 1.3.x with mod_perl, mod_php, mod_python and other Apache modules. If you wish to enable SSL, I have also written sections on how to do so.

The method of installation I describe is the DSO method, which is newer and takes just a few minutes to get everything up and running. This is not to say that this method is a bad, quick ”hack”. It is the most common way of installation when dealing with Apache and it’s modules. This method is much better for troubleshooting and even easily upgradeable.

Many of you may be wondering why this document does not (yet) cover Apache 2.0. I feel that Apache 2.0, as it’s still in beta releases currently, is not stable for serious web servers. Once it is, this document will be upgraded.

If you run across any problems, or if there are sections that you feel I should add, please email me at afra _at_ prongs.org .

2 Change log

v.5.00 (Changes from v.4.10)

- Added section on generating SSL certificates.
- Added configuration tips.
- Upgraded mod_perl section.
- Updated installation of IMAP libraries with PHP.
- Updated PHP installation method.

v.4.10 (Changes from v.3.50)

- Detailed DSO methods for building modules.

v.3.50 (Changes from v.3.00)

- Updated PHP Section
- Described how to compile IMAP support for PHP.

v.3.00 (Changes from v.2.55):

- Added mod_python 2.7.x section.
- Upgraded section of PHP to the PHP 4.1.x series.
- Expanded more on some sections.
- Spelling, spleling and speling.

v.2.55 (Changes from v.2.0):

- Rewrote document in Latex - PDF now available.
- Upgraded the section on PHP.

v.2.0 (Changes from v.1.5):

- Updated PHP notes to explain how to compile with GD and BCMath.
- Some spelling mistake fixes.

3 The Base Directory

Building all these packages can get messy. When one gets back to upgrading a module with Apache (or Apache itself), it would make sense to be able to find all the packages instantly, instead of trying to figure out which directory the modules were installed in.

I recommend to create a directory where all the packages/modules are to be downloaded to, untarred and built from. For example, on a Linux system, one may create the directory */usr/local/src/apache, or even /opt/apache*. On a Sun Server, I usually create the directory */export/apache*.

Now, it is under these directories where you should install all the packages/modules. Basically make a *base directory* that will be used to untar the packages needed to install Apache (mod_perl, mod_php, etc, etc). So a final look of the Apache build directories would look like (if one were to chose */usr/local/src/apache/* as the directory which will hold all the source files):

```
/usr/local/src/apache/apache_1.3.x
/usr/local/src/apache/mod_perl-1.2x
/usr/local/src/apache/mod_python-2.7.x
/usr/local/src/apache/php-4.3.x
/usr/local/src/apache/openssl-0.9.7d
...
```

Therefore, when it comes to upgrade time, one can simply find out what is installed by viewing the contents of the base directory that everything was originally installed under.

Note: throughout the document, I will refer to my base directory as /usr/local/apache. This can be different depending on what one chooses, for example, some choose to download the source files to */opt/apache/*.

I also chose to install Apache itself, after the configuration and compiling, in */usr/local/apache*. Now, this should not be confused with the source directories which were discussed above, as */usr/local/apache* should hold just the Apache server software itself, after it has been installed.

4 Download the software

Detailed in this section is where to obtain the software required.

4.1 Downloading the Modules

From the official Apache website, *http://www.apache.org*, you can download the source tarballs for Apache itself. For PHP, visit their official site *http://www.php.net*, and for mod_perl, *http://perl.apache.org*.

After downloading the different packages, I extracted them within my base directory, */usr/local/src/apache*.

5 The Apache Source

Once the source for Apache 1.3.x, one may untar it as shown:

```
# cd /usr/local/src/apache
# tar zxvf /path/to/apache-1.3.x.tar.gz
```

6 Package Configuration Tips

This section describes some tips one may use when configuring their modules.

6.1 Storing configure Parameters

Not only is keeping the directory for installing all the modules from in one place, important, but, so is how each package was configured. One may not remember how some configuration parameters were passed to the module in question in the future. Usually, in the module directory where the configure script was run from, there is a file called *config.status* which specifies which options were passed to the configure script. This file hould be kept in some other directory for safe keeping. Sometimes, however, this file may not be present after a build. Hence, it is always best to save the configuration parameters into a text file, named, for example *configure-parameters-YYYYMMDD.txt*.

6.2 Checking for configure Errors

When it comes to building the packages with the configure script, output is usually directed to the terminal window, and can be at a very fast pace. If there is an error or warning that the configure script indicates, one may not see what it is exactly, as the information has already scrolled down.

Therefore, I suggest saving the configure script's output to a text file, using the utility *tee*. As the man page for this utility states:

```
tee - read from standard input and write to standard output and files
```

One may test out tee by running the following from the prompt:

```
# ls | tee directory_listing.txt
```

One will not only see the contents of the directory on the screen but it will also be stored in the file, *directory_listing.txt*.

For each configure script detailed in this document, it is usually best to pipe *tee configure_output.txt*, and after the configure script is finished, to double check the file *configure_output.txt* to verify that everything went through okay. I have used this method throughout the configuration examples that are listed.

7 The SSL Packages

This section explains what should be downloaded and configured in order to have SSL integrated into Apache.

As mod_ssl applies patches to the Apache source tree, this *step should be carried out immediately after Apache has been untarred* if SSL is to be integrated. Otherwise, if SSL need not be installed, it is okay to skip the next section.

7.1 Download

If SSL is to be integrated with the Apache build, mod_ssl should be downloaded from *http://www.modssl.org*. Though, a system may already have OpenSSL, it is always wise and secure to check for the current releases. If OpenSSL needs to be upgraded and installed, the steps are shown below:

One will also need the mm package, which is available at *http://www.engelschall.com/sw/mm/*.

7.2 Configuring the packages

To integrate SSL into your web server, mm, mod_ssl and OpenSSL will have to be configured and built.

After downloading the tarballs as explained above, one may proceed to build mm (a memory management tool):

```
# cd /usr/local/src
# tar zxvf mm-1.2.2.tar.gz
# cd mm-1.2.2
# ./configure --disable-shared
# make
# make install
```

Now, onto OpenSSL, which should only be done if one does not have a recent version.

```
# cd /usr/local
# tar zxvf openssl-0.9.7d.tar.gz
# cd openssl-0.9.7d
# sh config no-idea -fPIC
# make depend [unless it does not ask you to do this]
# make
# make test
```

OpenSSL does not have to be installed at this point, however, issuing the command *make install* will install the files to */usr/local/*.

Now, onto mod_ssl:

```
# cd /usr/local
# tar zxvf mod_ssl-2.8.x.tar.gz
# cd mod_ssl-2.8.x
# ./configure \
  --with-apache=../apache-1.3.x \
  --with-ssl=../openssl-0.9.7d \
  --with-mm=../mm-1.2.x \
  --enable-shared=ssl \
  --enable-rule=EAPI \
  | tee configure_output.txt
```

The *–enable-shared=ssl* option enables the building of mod_ssl as a DSO module for Apache, *libssl.so*.

The *-enable-rule=EAPI* is required to add PHP or mod_perl support into Apache with mod_ssl!

After configure is run, messages similar to the following will be displayed:

```
Configuring mod_ssl/2.8.16 for Apache/1.3.29
 + Apache location: ../apache_1.3.29 (Version 1.3.29)
 + MM location: ../mm-1.2.2
 + Auxiliary patch tool: ./etc/patch/patch (local)
 + Applying packages to Apache source tree:
.
.
.
      + SSL library version: OpenSSL 0.9.7d 17 Mar 2004
```

Do not proceed any further with mod_ssl just yet.

Unless Apache is upgraded, *apxs* will be used to upgrade the mod_ssl module. More on this can be read at the end of this document.

8 Configuring Apache

Apache is now ready to be configured, which is usually a procedure carried out before any compilation. Please read the appropriate section below, depending if SSL is to be included, or not.

8.1 Perl Thread issues

If mod_perl is to be installed (regardless of the SSL status), and if Perl, on the system, was installed with *-lpthread*, the following will have to be passed to the Apache configure script, that will be outlined in the next section:

```
# CFLAGS="-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64" \
    LIBS="-lpthread" \
    ./configure ........
```

One can always tell if they have a thread-enabled version of Perl by checking the version:

```
# perl -v

This is perl, v5.8.3 built for i386-linux-thread-multi
```

Without the mention of threads, it is alright to proceed as normal without mentioning the LIBS line as above.

8.2 Without SSL

Without SSL, Apache can be configured as such:

```
./configure\
          --enable-module=rewrite \
          --prefix=/usr/local/apache \
          --enable-module=so \
  | tee configure_output.txt
```

Where *prefix* indicates to configure where Apache should be installed to. This is usually */usr/local/apache*, but can be */opt/apache*.

8.3 With SSL

If mod_ssl was compiled from the steps above, the following should be issued to the Apache source tree:

```
# cd /usr/local/src/apache/apache-1.3.x
# SSL_BASE=../openssl-0.9.6x \
  EAPI_MM=../mm-1.2.1 \
  ./configure\
         --enable-module=rewrite \
         --prefix=/usr/local/apache \
         --enable-shared=ssl \
         --enable-rule=SHARED_CORE \
         --enable-rule=SHARED_CHAIN \
         --enable-module=so \
         --enable-module=ssl \
 | tee configure_output.txt
```

Where *prefix* indicates to configure where Apache should be installed. This is usually */usr/local/apache*, but can be */usr/local/apache-ssl*, */opt/apache-ssl*, etc.

The following should be seen in the output of configure, though version numbers may be different:

```
 o ssl_module uses ConfigStart/End
     + SSL interface: mod_ssl/2.8.16
     + SSL interface build type: OBJ
     + SSL interface compatibility: enabled
     + SSL interface experimental code: disabled
     + SSL interface conservative code: disabled
     + SSL interface vendor extensions: disabled
     + SSL interface plugin: Built-in SDBM
     + SSL library path: /home/afra/apache/openssl-engine-0.9.7d
     + SSL library version: OpenSSL 0.9.7d [engine]  5 Dec 2002
     + SSL library type: source tree only (stand-alone)
```

If messages similar are seen, Apache has been configured correctly. If not, one should backtrack through the last steps and correct the issue. If Apache could not find OpenSSL, the OpenSSL source may have to be compiled from scratch as outlined above.

8.4 Configuration Notes

Depending what the Apache server needs to have enabled, special configuration options may have to be passed. For example, if the module *auth_dbm* (which comes with the Apache source) needs to be built in, by default, it won't be.

To get a list of what Apache automatically builds in, run *./configure –help* and a list will be shown. For example:

```
[env=yes         example=no      expires=no     ]
```

indicates that *mod_env* is compiled into Apache by default, whereas *mod_example* and *mod_expires* are not. If you decide to enable mod_example, simply add: *–enable-module=example* to the configure script that you ran in the above step.

Likewise, to disable a particular module, simply add *–disable-module=name* to the configure script.

One should be okay without worrying about this, but if the version of Apache needs special configuration it will not hurt to go over the list. To enable all the modules, append *-enable-modules=all -enable-modules=max* to the configure arguments, though the chances of a compilation error to occur seem more likely, and one may go through a hassle to install requirements for a module that may never used.

9 Compiling Apache

Whether SSL has been enabled or not, the compilation steps are the same:

```
# make
```

If compilation fails, it may be due to missing libraries and/or missing source code from the SSL module above. Please check everything once more if this is the case.

Once compiled and if this Apache build is built with mod_ssl, issue *make certificate* so that some test certificates can be created.

We may now install Apache by issuing *make install*. The files should be installed to the location that was specified to the configure script.

One of the advantages of the DSO method, is that once Apache is installed, and we would like to add some more modules to it in the future, the re-install of Apache does not have to occur! Through the aid of */path/to/installed/apache/bin/apxs*, one can easily build modules to add in a later time.

Congratulations! Apache has now been installed.

10 The SSL Certificates

This section is intended only for those that have installed Apache with SSL support, and describes how to enable SSL certificates within Apache.

10.1 Generating a Server Key

The first step is to create a 1024 bit RSA Private Key. Using OpenSSL, one can create the necessary Apache certificates. There are two different methods to do this: a server key with or without a password (called a pass phrase).

10.2 Password Enabled

```
# openssl genrsa -des3 1024 > server.key
Generating RSA private key, 1024 bit long modulus
.................++++++
.................++++++
e is 65537 (0x10001)
Enter pass phrase:
Verifying - Enter pass phrase for server.key:
```

You will have to enter a *pass phrase*, or password. *Do not lose this password!* Doing so will make your certificates useless! The password is case-sensitive and should be a combination of both letters and numbers. It is a good idea to backup these files on another form of media.

10.3 Using Expect

Everytime Apache is started, the pass phrase that you entered will have to be re-entered. If you do not wish to have a password with your certificate, read on the section below.

It is possible to have an Expect script start the Apache server by automatically entering in the password, for example:

```
#!/usr/bin/expect

spawn /usr/local/apache/bin/apachectl startssl
set password SSL-Certificate-Password
expect "pass phrase:"
send "$password\r"
expect eof
```

This is not advisable as the password is out there in plain text. If you do follow this method, it is strongly encouraged to allow *only* the root user to edit and execute it!

10.4 Without a password

Everytime Apache is started up, one will be asked for their pass phrase. This not necessarily convenient, as someone will have to be around everytime Apache is started up. Supposing the server is rebooted for whatever reason - Apache will have to be restarted manually. To get around this, one can create a Server Key, just like the sectiona above showed, however, Openssl will not ask for a pass phrase. This way, the pass phrase will not need to entered upon every Apache startup.

```
# openssl genrsa 1024 > server.key
```

One *must* make sure the server.key is only readable by the root user.

10.5 Viewing the Server Key

A confirmation that the server key was written correctly may be done via:

```
# openssl rsa -noout -text -in server.key
```

This will output details of the key that was just created.

10.6 Generating a Certificate Request

Now that a private key for the server has been saved, the next step is to generate a certificate request that one may send to any Certificate Authority, or CA. If there is no need for a CA to sign the certificate, please read the next section on Generating a Self-Signed certificate.

One will be asked a series of questions which contribute to the X.509 attributes of the certificate.

```
# openssl req -new -key server.key -x509 -days 365 -out server.csr

Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
--Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Illinois
Locality Name (eg, city) []:Chicago
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Some Company
Organizational Unit Name (eg, section) []:IT Department
Common Name (eg, YOUR name) []:www.servername.com
Email Address []:admin@servername.com
---
```

It is absolutely essential that one specifies the domain name that the certificate is being created for under **Common Name (eg. YOUR name)** (for example, *www.servername.com*).

The *challenge password* need not be entered - this is entirely optional.

Once completed, the file which has been created, *server.csr* should be sent to any Certificate Authority. Their websites will provide the details on how to do so.

The CA will then send you a file back, which should be saved as *server.crt*. It takes an average of a week or two for the CA to complete their process. In the meantime, one can generate a self-signed certificate for testing. Do not lose the file that the CA returns! Keep all keys on safe media.

10.7 Generating a Self-Signed Certificate

If one does not want a CA to sign their certificate, or for testing purposes, a self-signed certificate can be generated. The disadvantage to this is that the end user, when visiting the SSL enabled site, will be warned by their browser that the certificate could not be verified. However, the end user will still be able to continue onto the secure site after bypassing the error. For personal sites, or internal sites, this method is the cheapest and works.

The following will generate a certificate valid for 1 year and place the resulting certificate in the file, server.crt.

```
# openssl req -new -key server.key -x509 -days 365 -out server.crt
```

As in the above section, one will have to enter in details about the security certificate. Again, it is absolutely essential that one specifies the domain name that the certificate is being created for under **Common Name (eg. YOUR name)** (for example, *www.servername.com*). Again, keep these keys in a safe location, in addition to the server itself.

10.8 Another Similar Method

Here are basically the steps covered. If you have followed the above sections, this section can be skipped, though detailed is another (similar) method:

```
# openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.........++++++
...................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:

# openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Illinois
Locality Name (eg, city) []:Chicago
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SomeCompany
Organizational Unit Name (eg, section) []:IT Dept.
Common Name (eg, YOUR name) []:www.servername.com
Email Address []:admin@servername.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

One should send the resulting server.csr file to the CA, and they will send back a server.crt file.

To self sign the certificate:

```
# openssl x509 -req -days 360 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=/C=US/ST=Illinois/L=Chicago/O=SomeCompany/OU=IT Dept./CN=www.servername.com/emailAddres
Getting Private key
Enter pass phrase for server.key:
```

As with the above section, the files server.csr and server.key will be needed for Apache.

10.9 Installing the SSL Certificate

Whichever method one choses to generate the certificates, there should be two files *server.key* and *server.crt* (which is either the certificate sent from the CA, or the self-signed certificate).

One may place the server.key and server.crt files into any directory, however, the directories usually are */usr/local/apache/conf/ssl.key* and */usr/local/apache/conf/ssl.crt* respectively.

To install the SSL Certificates into Apache, one simply needs to notify Apache to where the certificate key files can be found on the server.

Within */path/to/installed/apache/conf/httpd.conf* (usually */usr/local/apache/conf/httpd.conf*), simply search for these directives, uncomment them and edit the file locations (if need be):

```
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt
```

11 Confirming the Apache Install

A test should be carried out to make sure Apache has been correctly installed. Fire up your favourite text editor and open up the file, */path/to/installed/apache/conf/httpd.conf*. For example, */usr/local/apache/conf/httpd.conf*.

Search for the following:

```
#Listen 3000
#Listen 12.34.56.78:80
```

The *Listen* directive notifies the Apache server of the IP address address and port number that it should listen on for request. For SSL requests, this is usually port 443. However, if mod_ssl was installed, something similar will be within the httpd.conf file:

```
<IfDefine SSL>
Listen 80
Listen 443
</IfDefine>
```

One should leave those lines commented out, but also add *Listen 127.0.0.1:80* directly underneath. Also, one should change the *ServerName* directive to *ServerName 127.0.0.1*.

127.0.0.1 can be changed to an IP address to any IP address from the network the server is on. However, it is always easiest to use 127.0.0.1 to test the Apache server install, as one can confirm Apache is running on the server itself, instead of running into any network configuration problems, if any.

11.1 Starting the Apache Server

Once *httpd.conf* has been saved, Apache can be started via:

```
# /usr/local/apache/bin/apachectl start
```

or if Apache is configured for use with SSL (the password for the certificate will be asked if one configured it so):

```
# /usr/local/apache/bin/apachectl startssl
```

If no problems are reported, Apache will inform you that the service has started. In any browser on the same computer, visit the url *http://127.0.0.1*, or *https://127.0.0.1*, and a message should appear that Apache has been installed successfully.

Once started one may check the *Error Log* which will indicate what has been compiled into Apache. For example:

```
# tail -f /path/to/installed/apache/logs/error_log

Apache/1.3.29 (Unix) mod_ssl/2.8.11 OpenSSL/0.9.7d mod_perl/1.29 PHP/4.3.5 configured
-- resuming normal operations
```

Another method is to also check the installed modules with Apache. This can be done by executing */path/to/installed/apache/bin/httpd -l*. One will see, amongst other modules, *mod_so* listed, and if SSL is enabled, *mod_ssl*.

At this point, the base Apache server has been installed. The next few sections discuss Apache add-on modules, and how to configure and install them.

12 The PHP 4.3.x module

PHP is a very popular Apache module. You can basically embed PHP code into HTML using special tags. For more information on PHP, visit their homepage at *http://www.php.net*.

There are many options used to build PHP. For example, what type of database the system has. The options are beyond the scope of this document. Please read the INSTALL file that comes with the PHP tarball.

There are many additional tools you will have to install to enable certain features in PHP. For example, if you have to program with graphic files, I do recommend you to install GD (graphics library) with PHP4.

Untar the PHP package:

```
# cd /opt/apache
# tar zxvf php-4.2.x.tar.gz
# cd php-4.2.x
```

12.1 IMAP Support

Due to the increased demand in webmail clients, I thought it would be appropriate to explain how one can configure PHP with IMAP support.

Download the IMAP package from *ftp://ftp.cac.washington.edu/imap/imap/* (you will want the latest version of imap - I downloaded *imap-2002e.tar.Z*).

After downloading the file, you will want to untar the package within your working base directory (mine was */usr/local/src/apache*). After untarring the package, you should change the directory to the subdirectory imap-2002e that was created.

Once in the subdirectory, we have to notify the make file what type of operating system we are on. Read the Makefile file, and you should see your OS listed. Using the code for your operating system, you will want to run:

If the IMAP library does not have to be compiled with OpenSSL libraries it is best to append *SSLTYPE=none* at the end of the make command.

Otherwise, if the IMAP compilation should be linked against OpenSSL, before compiling, it is wise to double check where the compiler will look for the OpenSSL libraries. One can change this in the file, *imap-2002e/src/osdep/unix/Makefile*. The values for the parameters *SSLDIR* should be changed, for example, to the path of the OpenSSL directory that was installed to from before, or to the default current location that exists on the server.

```
# make [operating system code] [optional: SSTYPE=none, if there is no need for  SSL].
```

For example, for FreeBSD, you would want to run *make bsf*. For Linux, try *make slx* (there are various options for Linux, though, so make sure you read the Makefile list properly).

Do not bother with make install, or if you want, you may go ahead and install the library and header files on your system. We will simply tell PHP to look in the current imap-2002e directory for the appropriate files.

After compiling, it is necessary to create some soft links for PHP's sake:

```
# cd /usr/local/src/apache/imap-2002e
# ln -s c-client lib
# ln -s c-client include
```

12.2 Configuring PHP

In my example, I will set PHP 4.3.x up with PostgreSQL and MySQL. One should untar the package as shown above into the base install directory (ie. */usr/local/src/apache*). Once untarred, we PHP is now ready to be configured.

12.3 Without SSL

The following is a sample configuration of PHP. It is not unusual to have different configuration script parameters, as this depends on the system and options needed for PHP. This configure script format should be followed either if one does not have mod_ssl installed, or if mod_ssl was configured with *–enable-rule=EAPI* (as detailed above).

```
# cd /opt/apache/php-4.3.x
# ./configure \
   --with-apxs=/path/to/installed/apache/bin/apxs \
   --with-pgsql=/usr/local/pgsql/ \
   --with-mysql=/usr/local/mysql/ \
   --enable-track-vars \
   --enable-versioning \
   --with-imap=../imap-2002e/ \
   --with-openssl=../openssl-0.9.7d/ \
   | tee configure_output.txt
```

The file *configure_output.txt* should be double checked to see if the configure script reported errors or not. If so, it is best to install or fix the problems detailed, and to proceed with configure again.

12.4 With SSL

If mod_ssl was compiled with the *–enable-rule=EAPI*, this section should be ignored and PHP should be configured outlined above (compiling PHP without SSL support).

Otherwise, there is a slight difference when compiling PHP. Prior to the configure script call, PHP should be notified to be built with *DEAPI*, as follows:

```
# cd /usr/local/src/apache/php-4.3.x
# CFLAGS='-DEAPI' ./configure \
   --with-apxs=/path/to/installed/apache/bin/apxs \
   --with-pgsql=/usr/local/pgsql/ \
   --with-mysql=/usr/local/mysql/ \
   --enable-track-vars \
   --enable-versioning \
   --with-imap=../imap-2002e/ \
   --with-openssl=../openssl-0.9.7d/ \
   | tee configure_output.txt
```

One may also want to add other CFLAGS parameters. For example, under Linux, it may be best to also add *-O2*, as such: *CFLAGS='-02 -DEAPI'*.

The file *configure_output.txt* should be double checked to see if the configure script reported errors or not. If so, it is best to install or fix the problems detailed, and to proceed with configure again.

12.5 Configuration Notes

Additional interesting options one may add to the *./configuration* script include *—with-java=DIR*, *—with-ldap=DIR*, *–enable-ftp*, *—enable-trans-sid*, and so on. Obviously each installation of PHP caters to the different needs for different servers. One may view the additional options by *./configure —help*.

A warning about databases and PHP, with mod_perl. I recently came across what seemed like a normal installation of Apache, PHP and mod_perl, however, the Apache server kept terminating itself without warning. Upon investigation, I found that PHP was using PHP's default MySQL libraries, and not the official MySQL libraries that were present on the server. This caused a big conflict between PHP and mod_perl. If you are going to build PHP with MySQL, you have to configure it properly by adding this option to the configuration script: *—with-mysql=/usr/local/*. In this case, the directory */usr/local/* contains the external MySQL libraries, and therefore PHP used those and not the internal libraries.

The *—with-apache* tag shows PHP where the Apache source code is (from above). You may even use the full path name here (ie. */usr/local/src/apache/apache_1.3.x*). If PHP complains that PostgreSQL could not be found, you will have to point out to configure where the header files are for PostgreSQL. The same rules apply if you are using Oracle or MySQL instead of PostgreSQL (read the README file for the specifics for the different database systems).

You will see a bunch of configuration values whiz by informing you what PHP has found on your system. The section, Configuration Tips, will explain how one can save these messages to a text file.

12.6 Installing PHP

After the configure is set, one will now have to compile the source code by typing **make** at the prompt. After the make is complete, typing *make install* will install the libraries and other required files. One may confirm that the files have

been installed correctly in the Apache directory tree by checking the directory (*/path/to/apache/libexec/libphp4.so*).

12.7 Setting up PHP

To inform Apache to load mod_php at startup time, *httpd.conf* must be edited and the following must be typed in:

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

underneath the *Dynamic Shared Object (DSO) Support* section. PHP may have already done this for you. Also, Apache must know when to use the mod_php module, therefore you will need to add this in the relevant section:

```
AddType application/x-httpd-php .php .phtml .php4 .php3
```

One may also wish to edit the *DirectoryIndex* directive in httpd.conf to:

```
DirectoryIndex index.html index.php index.phtml index.php4 index.php3
```

At this point, everything is all set. Restart Apache and view the error log to confirm the installation was a success.

12.8 Confirming PHP Installation

The *htdocs* directory in an Apache installation, is the directory from which Apache will serve the webpages from. This is probably */path/to/apache/htdocs*, or */usr/local/apache/htdocs*.

By placing the following text, in a file (preferably called *test.php*), within the htdocs directory, one can verify the installation of PHP.

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

By viewing the file through Apache, the browser must be pointed to *http://127.0.0.1/test.php*. One will see the different configuration parameters that were passed to the PHP configure script, as well as many other details.

13 The mod_python module

Python is yet another wonderful scripting language. It is object oriented, and is also very fast at text processing.

Firstly, one should make sure that the system has Python 2.x installed. Typing *python*, or *python -V*, at your command prompt should confirm this. If not, one will need to download and install Python. Python can downloaded from *http://www.python.org*. It is recommended to download a release from the Python 2.x series.

For example, this is what is seen when I type python at the prompt:

```
Python 2.3.3 (#2, Feb 24 2004, 09:29:20)
[GCC 3.3.3 (Debian)] on linux2
```

Even though one may have Python installed, one should install it **without** thread support, for the sake of mod_python.

If your version of Python was compiled with threads, it is best to install another copy of Python on the system without support for threads.

As for mod_python itself, one can download a copy from *http://www.modpython.org*. Shown in this document is how to install the mod_python 2.7.x series. After obtaining the package, untar the file:

```
# cd /opt/apache
# tar zxvf /path/to/mod_python-2.7.x
# cd mod_python-2.7.x
```

13.1 The mod_python module

If one did not set anything up for PHP, in the previous section, you will need to configure Apache first. Simply follow the instructions from the previous section in this document, *Pre-configuring Apache*. If, however, you have already done this for PHP, you need not re-configure Apache.

We are now ready to configure mod_python. I did so as follows:

```
# ./configure --with-apache=../apache_1.3.x
```

It should be noted that if you had to install another version of Python, to avoid confusion for mod_python, you will have to specify where the source is for the Python you wish to use. You may do this as follows:

```
# ./configure --with-apache=../apache_1.3.x \
              --with-python=/path/to/Python
```

You can also specify where, if any, databases are installed on your system.

After the configure is finished, proceed to make mod_python by issuing *make* command. After the compilation, install mod_python with *make install*. The mod_python files will be installed in the *src/modules/python* subdirectory within your Apache source directory.

14 The mod_perl module

mod_perl is a powerful language that one can use with Apache and is better to use than plain Perl for CGI purposes in many instances (speed and memory being important ones).

14.1 Configuring mod_perl

Here's how one would set up mod_perl for the build:

```
# cd /opt/apache
# tar zxvf mod_perl-1.2x.tar.gz
# cd mod_perl-1.2x
```

mod_perl should be configured with a command similar to:

```
# perl Makefile.PL USE_APXS=1 WITH_APXS=/path/to/apache/bin/apxs EVERYTHING=1
```

Note to SSL users: If one configured and installed mod_ssl with the *–enable-rule=EAPI* option, then the above is fine. If not, one will have to enable compilation with the CFLAGS set to *-DEAPI* as described in the PHP section.

If the configuration of mod_perl complains of missing Perl modules, one can obtain them from *http://www.cpan.org*. Build and install the modules, and then restart the mod_perl build. Each module that may be required will have specific install instructions, usually within an INSTALL file. Usually the process for installing a Perl module is:

```
# perl Makefile.PL
# make
# make install
```

14.2 Installing mod_perl

If the configuration went smoothly, one may go ahead and proceed with the installation of mod_perl:

```
# make
# make test
# make install
```

The resulting file will be called *libperl.so* and is installed into */usr/local/apache/libexec* (or wherever Apache was installed to).

14.3 Setting up mod_perl

Once all files have been installed, Apache will need to be notified about the new modules. mod_perl may have already done this, however, it is always best to check. Within the file */path/to/apache/conf/httpd.conf*, the following will have to be added if it is not already present:

```
LoadModule perl_module        libexec/libperl.so
```

More can be read on setting up mod_perl further on the mod_perl homepage.

14.4 Additional Perl Modules

Powerful, flexible and effecient modules can be downloaded to ease development with mod_perl. For example, the *Mason::HTML* module can enable Apache to parse mod_perl within HTML pages

15 Verification

One should check that Apache has been installed properly. Apache's configuration file in */path/to/apache/conf/httpd.conf* should be read through. The ServerName, ServerAdmin entries should probably be changed. This configuration file will be where one can alter Apache's behaviour.

Once Apache's configuration has been read and confirmed, the Apache server should be started up. Make sure no other Apache processes exist (if they do issue a *killall apache*). To start it up you would want to:

```
/usr/local/apache/bin/apachectl start
```

Now Apache has started confirm the installed modules by tailing the error log file:

```
tail -f /usr/local/apache/logs/error_log
```

and you should see something similar to:

```
Apache/1.3.x (Unix) PHP/4.3.x mod_perl/1.2x mod_python/2.7.x Python/2.2.x
mod_ssl/2.8.x OpenSSL/0.9.7d configured
```

(obviously the result will depend on the module(s) you installed). Another trick to see which modules have been installed is:

```
/usr/local/apache/bin/httpd -l
```

To put a bigger smile on your face, visit *http://www.netcraft.com/whats/?host=www.yoursite.com*.

15.1 Upgrading in the future

Once Apache is built, modules can be added and removed from *httpd.conf*, and the installation of modules is independant of the Apache install, as shown when installing PHP or mod_perl.

That is why DSO is so flexible as it is easier to manage Apache packages.

16 Links

- *http://www.apache.org* - The homepage for the Apache web server.
- *http://perl.apache.org* - The homepage for mod_perl.
- *http://www.php.net* - The homepage for PHP.
- *http://www.python.org* - The homepage for mod_python.
- *http://www.openssl.org* - The OpenSSL project.
- *http://www.modssl.org* - The mod_ssl homepage.
- *http://www.engelschall.com/sw/mm/* - Homepage for mm.
- *http://www.cpan.org* - Archive of Perl modules.
- *http://www.masonhq.com* - HTML::Mason.