

**In this chapter:**

- Sample Installation
- Configuration Reference
- Configuration Special Topics

## Chapter 3

# Building Apache



*The software said it requires Microsoft IIS 4 or better, so I installed Apache.*

— Unknown (paraphrased)

In this chapter, we first discuss the process of building a full-featured Apache web server with Perl scripting and SSL/TLS capabilities; our discussion takes the form of a step-by-step tutorial that introduces the *Apache AutoConf-style Interface* (APACI). The procedure includes getting and extracting the distributions, configuring the source trees, building the packages, and finally installing the packages. A complete reference follows the tutorial and covers all command-line variables and options of APACI in detail. Selected Apache configuration special topics are also discussed in more detail at the end of the chapter.

### 3.1 Sample Step-by-Step Installation

In the first part of this chapter, we show a complete step-by-step installation procedure for an Apache server including two extensions, `mod_perl` and `mod_ssl`. These two particular modules were selected because they are also covered in Chapter 4 (the configuration chapter). Other popular third-party modules such as `mod_php3`, `mod_dav`, and `mod_jserv` can be added in an analogous fashion. You can therefore treat this section as an installation tutorial

Here we build a full-featured Apache web server with Perl scripting and SSL/TLS capabilities.

intended to help you better understand the configuration reference in Section 3.2 and to get a general impression of how to perform a complex Apache installation.

### 3.1.1 File System Preparation

As the first step, you must decide where the server package should be installed. That is, you have to choose a common installation path prefix, such as `/usr/local/apache`. This prefix is important because it is required repeatedly in the installation procedure. All packages will be installed in subdirectories under this prefix. The file system on which this path prefix will reside must have at least 25MB of free disk space available. In addition, you need a working directory where the packages can be built prior to the installation. For this temporary area, additional disk space of at least 60MB is required.

For installing Apache you need 25MB free disk space for installation plus 60MB temporary disk space.

### 3.1.2 Obtaining the Source Distribution

Next, you must obtain the Apache source distribution. Although so-called binary distributions are available, we recommend that you start with the source distribution, except in those very rare cases where no C compiler is available. First, determine the latest version number of Apache by looking at the Apache home page (<http://www.apache.org/httpd>). It always includes a note about the latest version.

Package	Source Distribution
Apache	<a href="http://www.apache.org/dist/apache_1.3.12.tar.gz">http://www.apache.org/dist/apache_1.3.12.tar.gz</a>
mod_perl	<a href="http://perl.apache.org/src/mod_perl-1.24.tar.gz">http://perl.apache.org/src/mod_perl-1.24.tar.gz</a>
Perl	<a href="http://www.perl.com/cpan/src/5.0/perl-5.005_03.tar.gz">http://www.perl.com/cpan/src/5.0/perl-5.005_03.tar.gz</a>
mod_ssl	<a href="http://www.modssl.org/source/mod_ssl-2.6.6-1.3.12.tar.gz">http://www.modssl.org/source/mod_ssl-2.6.6-1.3.12.tar.gz</a>
OpenSSL	<a href="http://www.openssl.org/source/openssl-0.9.6.tar.gz">http://www.openssl.org/source/openssl-0.9.6.tar.gz</a>

**Table 3.1:** The involved software packages

For instance, at the time of this writing, the most recent Apache version was 1.3.12. To grab the source distribution, you can use a browser and fetch it via HTTP from the area <http://www.apache.org/dist/>. Apache is also available from mirror locations around the world. The list of available locations can be found at <http://www.apache.org/mirrors/>.

In this book, we also cover two popular Apache extensions, `mod_perl` (the Apache interface to Perl) and `mod_ssl` (the Apache interface to OpenSSL). You must therefore obtain the distributions of four additional packages: the Perl interpreter ([www.perl.com](http://www.perl.com)), the `mod_perl` module ([perl.apache.org](http://perl.apache.org)),

All of these software packages are available free of charge on the Internet.

the OpenSSL toolkit ([www.openssl.org](http://www.openssl.org)), and the `mod_ssl` module ([www.mod-ssl.org](http://www.mod-ssl.org)).

Table 3.1 on the preceding page summarizes the download locations for the latest stable version of all involved packages. If they have changed since this book's publication, start over with the home page of the package to find the latest version and current download location.

To build the packages, move the distribution files into your chosen working directory and extract the five distribution “tarballs” there.

```
$ gunzip -c apache_1.3.12.tar.gz | tar xvf -
apache_1.3.12/
apache_1.3.12/src/
apache_1.3.12/src/ap/
:
$ gunzip -c mod_perl-1.24.tar.gz | tar xvf -
mod_perl-1.24/
mod_perl-1.24/t/
mod_perl-1.24/t/docs/
:
$ gunzip -c perl-5.005_03.tar.gz | tar xvf -
perl5.005_03/
perl5.005_03/Artistic
perl5.005_03/Changes
:
$ gunzip -c mod_ssl-2.6.6-1.3.12.tar.gz | tar xvf -
mod_ssl-2.6.6-1.3.12/ANNOUNCE
mod_ssl-2.6.6-1.3.12/CHANGES
mod_ssl-2.6.6-1.3.12/CREDITS
:
$ gunzip -c openssl-0.9.6.tar.gz | tar xvf -
openssl-0.9.6/CHANGES
openssl-0.9.6/CHANGES.SSLeay
openssl-0.9.6/Configure
:
```

### 3.1.3 Package Prerequisites

The Apache build process discussed later in this chapter depends on the availability of the `mod_perl` and `mod_ssl` modules. But these, in turn, depend on the Perl and OpenSSL third-party packages. For this reason, a prerequisite to building Apache with these modules is to build these two packages first. Follow these steps to install Perl and OpenSSL:

- Go into the source tree of Perl and follow the directions in the `INSTALL` document found there. The typical installation steps will look like the following:

To build the complex Apache modules, the third-party packages must usually be installed first.

```

$ cd perl-5.005_03
$ sh Configure -d -s -e -Dprefix=/usr/local/apache
First let's make sure your kit is complete.  Checking...
Locating common programs...
:
$ make
'sh cflags libperl.a miniperlmain.o' miniperlmain.c
: CCCMD = cc -DPERL_CORE -c -I/usr/local/include -O
$ make install
./perl installperl
mkdir /usr/local/apache/bin
:
$ (cd /usr/include; /usr/local/apache/bin/h2ph *.h sys/*.h machine/*.h)
a.out.h -> a.out.ph
acl.h -> acl.ph
:
$ cd ..

```

- Go into the source tree of OpenSSL and follow the directions in its INSTALL document. The typical installation steps will be similar to the following:

```

$ cd openssl-0.9.6
$ sh config --prefix=/usr/local/apache
Operating system: i386-whatever-freebsd3
Configuring for FreeBSD-elf
:
$ make
making all in crypto...
echo "#define DATE ``date``" >date.h
gcc -I. -I../include -DTERMIOS -DL_ENDIAN ...
:
$ make install
installing crypto...
making install in crypto/md2...
:
$ cd ..

```

After you complete these steps, the Perl interpreter and the OpenSSL toolkit will be installed under `/usr/local/apache`. Now, you must proceed with the application of `mod_perl` and `mod_ssl` to the Apache source tree.

Follow these steps:

- Go into the source tree of `mod_perl` and follow the directions in the `INSTALL.apaci` document found there. Many configuration options are available for `mod_perl`, although the typical installation steps will look like the following:

Good packages usually do not require you to manually edit their configuration. Instead, they provide some sort of autoconfiguration mechanism.

Complex Apache modules usually provide automated ways to get to the Apache source tree. Most rely on APACI features.

```

$ cd mod_perl-1.24
$ /usr/local/apache/bin/perl Makefile.PL APACHE_SRC=./apache_1.3.12
DO_HTTPD=1 USE_APACI=1 PREP_HTTPD=1 EVERYTHING=1
Will configure via APACI
(cd ../apache_1.3.12/src && ./Configure -file Configuration)
:
$ make
mkdir blib
mkdir blib/lib
cp ../apache_1.3.12/src/include/http_protocol.h ...
:
$ make install
Installing /usr/local/apache/lib/perl5/site_perl/5.005/...
:
$ cd ..

```

- Go into the source tree of `mod_ssl` and follow the directions in its `INSTALL` document. Once again, many configuration options are available, but the typical installation steps will look like the following:

```

$ cd mod_ssl-2.6.6-1.3.12
$ ./configure --with-apache=./apache_1.3.12
Configuring mod_ssl/2.6.3 for Apache/1.3.12
+ Apache location: ../apache_1.3.12 (Version 1.3.12)
:
$ cd ..

```

### 3.1.4 Configuring the Apache Source Tree

The next major step is to configure the Apache source tree. You must select the desired modules, the compiler and flags used for building, the installation path layout, and other features. Typically, you will use the *Apache AutoConf-style Interface* (APACI) — a script named “`configure`” you will find in the top-level of the Apache source tree.<sup>1</sup> This script provides numerous options that allow you to flexibly configure the build and installation. A complete option reference appears in Section 3.2.

The minimal configuration step usually takes the following form:

```

$ cd apache_1.3.12
$ ./configure --prefix=/usr/local/apache

```

<sup>1</sup>In the configuration method of Apache 1.2, you had to manually edit a configuration file (`src/Configuration`) and run the script (`src/Configure`) on it. Because this older facility doesn't provide installation support, you are strongly advised to use APACI unless you are an Apache expert. Even when you are an Apache expert, you'll discover that these editing steps can be carried out via APACI's command-line options as well.

The recommended standard way to configure the Apache 1.3 source tree is the new Apache AutoConf-style Interface.

```
Configuring for Apache, Version 1.3.12
:
$ cd ..
```

Because we want to build Apache with our two additional modules and because it's reasonable to build Apache as flexibly as possible with the help of the *Dynamic Shared Object* (DSO) facility, we recommend the following steps:

APACI offers lots of command-line options, which at first look are ugly. Once you become familiar with APACI, you will enjoy its consistent and batch-capable nature.

```
$ cd apache_1.3.12
$ env SSL_BASE=/usr/local/apache ./configure
--target=apache
--with-layout=GNU
--without-confadjst
--prefix=/usr/local/apache
--with-perl=/usr/local/apache/bin/perl
--activate-module=src/modules/perl/libperl.a
--enable-module=perl
--enable-module=ssl
--enable-shared=remain
```

```
Configuring for Apache, Version 1.3.12
+ using installation path layout: GNU (config.layout)
+ activated perl module (modules/perl/libperl.a)
Creating Makefile
Creating Configuration.apaci in src
Creating Makefile in src
+ configured for FreeBSD 3.1 platform
+ setting C compiler to gcc
+ setting C pre-processor to gcc -E
+ checking for system header files
+ using custom target name: apache
+ adding selected modules
  o rewrite_module uses ConfigStart/End
    enabling DBM support for mod_rewrite
  o dbm_auth_module uses ConfigStart/End
  o db_auth_module uses ConfigStart/End
    using Berkeley-DB/1.x for mod_auth_db (-lc)
  o ssl_module uses ConfigStart/End
    + SSL interface: mod_ssl/2.6.3
    + SSL interface build type: OBJ
    + SSL interface compatibility: enabled
    + SSL interface experimental code: disabled
    + SSL interface vendor extensions: disabled
    + SSL interface plugin: Vendor DBM (libc)
    + SSL library path: /usr/local/apache
    + SSL library version: OpenSSL 0.9.6
    + SSL library type: installed package (stand-alone)
    + SSL library plugin mode: none
```

```

o perl_module uses ConfigStart/End
+ mod_perl build type: OBJ
+ id: mod_perl/1.24
+ id: Perl/5.00503 (freebsd) [/usr/local/apache/bin/perl]
+ setting up mod_perl build environment
+ adjusting Apache build environment
+ enabling Perl support for SSI (mod_include)
+ enabling Extended API (EAPI)
+ doing sanity check on compiler and options
Creating Makefile in src/support
:
```

These steps configure Apache as follows:

1. The `SSL_BASE` variable locates the installed OpenSSL package for module `mod_ssl`.
2. `--target` names the program “apache” (the default is “httpd”).
3. `--with-layout` selects a GNU-style file system layout (the default is an old-style “Apache” layout).
4. `--prefix` specifies the installation path prefix.
5. `--sbindir` ensures that all binaries are installed into a single directory for binaries.
6. `--with-perl` locates the installed Perl package for `mod_perl`.
7. `--activate-module` activates the `mod_perl` module for the configuration process.
8. The two `--enable-module` options enable `mod_perl` and `mod_ssl`.
9. `--enable-shared` forces all unenabled modules to be built as DSOs for later loading on demand.

APACI options are just what their name implies: optional. Don't get confused by the large number of available options – they are just available for fine-tuning, but are not mandatory.

### 3.1.5 Building and Installing Apache

The final step is to actually build the Apache package ingredients and install them under our selected installation path prefix. Because we used APACI, this task is easy and usually involves only two simple commands:

```

$ make
==> src
==> src/os/unix
gcc -c -I../.. -I../..os/unix -I../.. / ...
:
$ make install
==> [mktree: Creating Apache installation tree]
./src/helpers/mkdir.sh /usr/local/apache/bin
./src/helpers/mkdir.sh /usr/local/apache/bin
:
```

Of course, we used `mod_ssl`. Consequently, a X.509 server certificate and private key are required, though they can be dummy ones for testing purposes. The following steps are required in our situation:

```
$ make
==> src
==> src/os/unix
gcc -c -I../.. -I../..os/unix -I../.. /...
:
$ make certificate TYPE=dummy
SSL Certificate Generation Utility (mkcert.sh)
:
$ make install
==> [mktree: Creating Apache installation tree]
./src/helpers/mkdir.sh /usr/local/apache/bin
./src/helpers/mkdir.sh /usr/local/apache/bin
:
```

Installing a ready-to-run Apache web server with the help of APACI is just a matter of a few make commands.

*Voilà!* You have now successfully installed an Apache web server including `mod_perl` and `mod_ssl` under `/usr/local/apache`. Your reward is an out-of-the-box usable Apache web server including Perl scripting and SSL/TLS functionality. To verify that it works properly, fire up your new server from the root user account ...

```
$ su
Password:
# /usr/local/apache/sbin/apachectl startssl
/usr/local/apache/bin/apachectl startssl: httpd started
% $ ps -ax | grep -i apache
% 8593 ?? Ss 0:01.87 /usr/local/apache/sbin/apache -DSSL
% 8594 ?? S 0:00.00 /usr/local/apache/sbin/apache -DSSL
% 8595 ?? S 0:00.00 /usr/local/apache/sbin/apache -DSSL
% 8596 ?? S 0:00.00 /usr/local/apache/sbin/apache -DSSL
% 8597 ?? S 0:00.00 /usr/local/apache/sbin/apache -DSSL
% 8598 ?? S 0:00.00 /usr/local/apache/sbin/apache -DSSL
```

... and try to connect to it with your favorite browser via both HTTP and HTTPS through the URLs `http://localhost/` and `https://localhost/`. Both protocols should be usable.

## 3.2 Configuration Reference

The following is a complete reference to the APACI command line. The various APACI command-line variables and options enable you to adjust the way Apache is built and installed. The command line has the following general structure:

```
$ env [VARIABLE=value ...] ./configure [--option=value ...]
```

### 3.2.1 Configuration Variables

The following *VARIABLES* are taken from the APACI environment:

- **CC** C Compiler Program  
 Example: `CC="egcc"`  
 Specifies the C compiler program to use when building the Apache object files and executables. The default is platform-dependent — Apache selects the best choice when more than one compiler is installed on the platform. You can also use this variable to force the usage of a particular compiler, however.
- **CFLAGS** C Compiler Standard Flags  
 Example: `CFLAGS="-Wall -pendantic"`  
 Specifies standard flags for the C compiler. These flags will be used on the `CC` command lines. The exception is `"-I"`, which (for historical reasons) should be specified with `INCLUDES`.
- **OPTIM** C Compiler Optimization Flags  
 Example: `OPTIM="-pipe -O2"`  
 Specifies the C compiler optimization flags. These flags can also be specified via `CFLAGS`, but for historical reasons they have their own variable.
- **INCLUDES** C Compiler Include Flags  
 Example: `INCLUDES="-I/usr/local/include"`  
 Specifies additional include flags for the C compiler (`"-Idirectory"`). These flags can also be specified via `CFLAGS`, but for historical reasons they have their own variable.
- **LDFLAGS** Linker Standard Flags  
 Example: `LDFLAGS="-L/usr/local/lib"`  
 Specifies additional standard flags for the linker command. These flags are usually `"-Ldirectory"` flags intended to help the linker find third-party libraries.
- **LIBS** Linker Library Flags  
 Example: `LIBS="-ldb"`  
 Specifies additional library flags for the linker command. These flags are usually just `"-lname"` flags for linking Apache with third-party libraries.
- **CFLAGS\_SHLIB** Additional CFLAGS for DSO Building  
 Example: `CFLAGS_SHLIB="-fPIC"`

To squeeze out the maximum on a Pentium platform, compile with `CC=pgcc CFLAGS='-O6 -mpentium'` when the Pentium optimized GNU C compiler is available. This option speeds up at least number-crunching tasks such as regex matching.

In most cases, you do not need to specify DSO building details manually. Apache already knows how to use DSO on all major UNIX platforms.

Special flags for the C compiler that are used in addition to CFLAGS when DSOs are compiled. Usually, they specify which flags are required to force the generation of *position-independent code* (PIC).

■ **LD\_SHLIB** Linker for DSO Building

Example: LD\_SHLIB="ld"

Specifies the linker used for building DSOs. The default is platform-dependent and is usually either the value of CC or directly "ld".

■ **LDFLAGS\_SHLIB** Additional LDFLAGS for DSO Building

Example: LDFLAGS\_SHLIB="-Bshareable"

Special flags for the DSO linker command (LD\_SHLIB) that are used in addition to LDFLAGS when DSOs are built. Usually, they specify which flags are required to force the generation of shared objects instead of standard objects (executables).

■ **LDFLAGS\_SHLIB\_EXPORT** Add. LDFLAGS for Program Building under DSO

Example: LDFLAGS\_SHLIB\_EXPORT="-rdynamic"

Special flags for the linker command (CC) that are used in addition to LDFLAGS when the Apache executable is built. Usually, they specify which flags are required to force the export of API symbols for use by DSO-based modules.

■ **RANLIB** Archive Indexing Tool

Example: RANLIB="/bin/true"

Used to override the "ranlib" command if the local platform doesn't require it. This command is rarely needed, because Apache automatically knows whether it is required.

■ **DEPS** Additional Makefile Dependency

Example: DEPS="..."

For developers only. This command can be used to add a *Make* dependency to `src/Makefile`.

■ **TARGET** Name of the Target Program

Example: TARGET="apache"

Equivalent to the option `--target`. (See this option for details.)

■ **EAPI\_MM** Path to MM Library (mod\_ssl only)

Example: EAPI\_MM="SYSTEM"

Sets the path to the MM library source or installation tree. This library is used in conjunction with the *Extended API* (EAPI) facility. The argument "SYSTEM" can be specified to indicate that APACI should search for the MM library in standard system locations.

- **SSL\_BASE** Path to OpenSSL Toolkit (mod\_ssl only)

Example: `SSL_BASE="SYSTEM"`

Sets the path to the OpenSSL toolkit source or installation tree. This toolkit is used in conjunction with `mod_ssl`. The string "SYSTEM" can be specified as the argument to indicate that APACI should search for the OpenSSL toolkit in standard system locations.

`mod_ssl` extends APACI; that's why additional configuration variables are available.

- **RSA\_BASE** Path to RSAref Library (mod\_ssl only)

Example: `RSA_BASE="SYSTEM"`

Sets the path to the RSAref library source or installation tree. This library is used by U.S. residents (only) in conjunction with `mod_ssl`. The string "SYSTEM" can be specified as the argument to indicate that APACI should search for the RSAref library in standard system locations.

### 3.2.2 General Options

The following general *options* are available on the APACI command line:

- **--quiet** Quiet Configuration

Example: `--quiet`

Forces APACI to suppress the display of all output while configuring the source tree. This option can be useful for running APACI in batch mode — for instance, from a vendor packaging facility like RPM.

- **--verbose** Verbose Configuration

Example: `--verbose`

Forces APACI to display additional output while configuring the Apache source tree. This option can be useful for testing the results of various configuration options.

- **--shadow[=*Dir*]** Create a Shadow Source Tree

Example: `--shadow=/tmp/apache`

Creates a shadow tree of the Apache source tree under the specified directory *Dir* before configuration. A shadow tree consists of all directories of the original tree, with all files inside those directories being replaced by symbolic links to the original files. This option is useful when the original Apache source tree stays on a read-only medium (typically, a CD-ROM) or when one compiles Apache in parallel for multiple platforms. Details about this facility are found in section 3.3.1.

Use `--shadow` for building Apache on multiple architectures in parallel.

### 3.2.3 Stand-alone Options

The following *options* stand alone. That is, they stop the configuration process and instead perform a special action only.

- **--help** Display Usage Summary  
 Example: `--help`

Displays a short usage summary page listing all APACI command-line options. Use this option when you have forgotten an option and want to identify it.
- **--show-layout** Display Installation Path Layout  
 Example: `--show-layout`

Displays the resulting installation path layout. By default, this option displays the “Apache” layout from the file `config.layout`. It is also useful in combination with the `--with-layout` option and the other installation layout options (see section 3.2.4) for easily checking the results of those options without having to configure, build, and install the package.

### 3.2.4 Installation Layout Options

The following *options* are used for configuring the general installation path layout:

- **--with-layout=[File:]Name** Installation Path Layout  
 Example: `--with-layout=GNU`

Selects the predefined installation path layout named *Name* from the file `config.layout`. This file includes several popular layout ingredients, and you can set all of their paths at once with this single option. The default is the historical “Apache” layout. The most typical layout is “GNU,” which resembles the installation paths of typical GNU Autoconf-based packages. When the *Name* argument is prefixed with “File:”, *Name* is loaded from *File* instead of `config.layout`. Use this option for easy loading your own custom layouts.
- **--target=Name** Installation Target Name  
 Example: `--target=apache`

Sets the name of the target program to *Name*. The default is the historical “httpd”. This name affects both the name of the installed executable and the error messages.

The `--with-layout` option allows you to load custom installation path layouts from a file.

■ **--prefix=Prefix** Installation Path Prefix

Example: `--prefix=/usr/local/apache`

As with original GNU Autoconf “configure” scripts, this option is the most important choice. It sets the installation path prefix — that is, the root of the installation tree. Because most other installation path-related options are, by default, subdirectories of this path, this option implicitly changes the value of these options unless you configure them manually. The default for *Prefix* is `/usr/local/apache`. In most cases, this option is all you need to force the installation to take place in a different file system area.

The only mandatory APACI is `--prefix`.

■ **--exec-prefix=ExecPrefix** Installation Path Prefix for Executables

Example: `--exec-prefix=/usr/local/apache.'uname -m'`

Similar to `--prefix`, but configures only the prefixes for executables — or, more correctly, for architecture-dependent files. The default for *ExecPrefix* is *Prefix*; that is, by default APACI doesn't distinguish between the various types of files. This option is useful primarily when you install Apache for multiple architectures into the same installation area.

■ **--datadir=DataDir** Installation Path Prefix for Shared Data

Example: `--datadir=/usr/local/apache/share`

Sets the installation path prefix for the static, read-only data files used by Apache (such as hypertext documents, and CGI scripts). The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *DataDir* usually defaults to *Prefix*; that is, this directory is usually a path prefix for other paths.

■ **--localstatedir=StateDir** Inst. Path Prefix for Dynamic Data

Example: `--localstatedir=/usr/local/apache/var`

Sets the installation path prefix for the various dynamic/writable data files used by Apache (such as log files). The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *StateDir* usually defaults to *Prefix*; that is, this directory is usually a path prefix for other paths.

By default, APACI uses a three-step path dependency hierarchy: `--with-layout` at the top, then a few options like `--datadir` that group related files, and finally specialized options like `--bindir` for fine-tuning.

The following *options* are used for fine-tuning the installation path layout:

■ **--bindir=Dir** Installation Path for User Binaries

Example: `--bindir=/usr/local/apache/bin`

Sets the installation path for user binaries — that is, executables that will be run by the end user. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *ExecPrefix/bin*.

- **--sbindir=Dir** Installation Path for System Binaries  
 Example: `--sbindir=/usr/local/apache/bin`  
 Sets the installation path for system binaries — that is, executables that will be run by system administrators only. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *ExecPrefix/bin*.
- **--libexecdir=Dir** Installation Path for Internal Binaries  
 Example: `--libexecdir=/usr/local/apache/libexec`  
 Sets the installation path for internal binaries — that is, executables and DSOs that will be loaded and run by Apache itself. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *ExecPrefix/libexec*.
- **--mandir=Dir** Installation Path for Manual Pages  
 Example: `--mandir=/usr/local/apache/man`  
 Sets the installation path for the UNIX manual pages. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *Prefix/man*.
- **--sysconfdir=Dir** Installation Path for Configuration  
 Example: `--sysconfdir=/usr/local/apache/etc`  
 Sets the installation path for the various configuration files. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *Prefix/conf*.
- **--includedir=Dir** Installation Path for C Include Files  
 Example: `--includedir=/usr/local/apache/include`  
 Sets the installation path for the C language include files (also known as “header files”), which are used by the APXS facility. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *Prefix/include*.
- **--iconsdir=Dir** Installation Path for Icons  
 Example: `--iconsdir=/usr/local/apache/share/icons`  
 Sets the installation path for icon images. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *DataDir/icons*.
- **--htdocsdir=Dir** Installation Path for Hypertext Documents  
 Example: `--htdocsdir=/usr/local/apache/share/htdocs`  
 Sets the installation path for hypertext documents. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *DataDir/htdocs*.

Most of the APACI options are similar in name to the ones specified by the GNU standards.

- **--cgidir=*Dir*** Installation Path for CGI Scripts  
 Example: `--htdocsdir=/usr/local/apache/share/cgi-bin`

Sets the installation path for CGI scripts. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *DataDir/cgi-bin*.
- **--runtimedir=*Dir*** Installation Path for Runtime Data  
 Example: `--runtimedir=/usr/local/apache/var`

Sets the installation path for the runtime data of Apache (scoreboard, PID file, and so on). The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *StateDir/logs*.
- **--logfiledir=*Dir*** Installation Path for Log Files  
 Example: `--logfiledir=/usr/local/apache/var`

This sets the installation path for the log files. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *StateDir/logs*.
- **--proxycachedir=*Dir*** Installation Path for Proxy Cache  
 Example: `--proxycachedir=/usr/local/apache/var`

Sets the installation path for the proxy module's cache. The default depends on the installation path layout (see the discussion of the `--with-layout` option), but *Dir* usually defaults to *StateDir/proxy*.

### 3.2.5 Build Options

The following *options* are used for configuring the various build parameters:

- **--enable-rule=*Name*** Enable a Configuration Rule  
 Example: `--enable-rule=WANTHSREGEX`

Used to enable various configuration rules. The following rule *Names* are available:

  - SHARED\_CORE Configures the Apache core to be built into a shared library.
  - SHARED\_CHAIN Configures the linking of module DSOs against possibly existing shared libraries.
  - SOCKS4 Builds Apache with the SOCKS version 4 toolkit. When it is enabled, you must add the SOCKS library location to LIBS, otherwise, `"-L/usr/local/lib -lsocks"` will be assumed.

For configuring special details of Apache, some configuration rules exist. Some of them were added to APACI by `mod_ssl`.

**SOCKS5** Builds Apache with the SOCKS version 5 toolkit. When it is enabled, you must add the SOCKS library location to `LIBS`, otherwise, “`-L/usr/local/lib -lsocks5`” will be assumed.

**IRIXNIS** Takes effect only if you are configuring on SGI IRIX. Read the `src/Configuration.tmp1` file for more details.

**IRIXN32** Takes effect only if you are configuring on SGI IRIX. Read the `src/Configuration.tmp1` file for more details.

**PARANOID** During the configuration, modules can run pre-programmed shell commands in the same environment in which APACI runs. This rule allows modules to control how APACI works. Normally, APACI will simply note that a module is performing this function. If you use this rule, it will also print out the code that the modules execute.

**EXPAT** Includes James Clark’s *Expat* package (an XML/1.0 parsing library) into Apache, for use by the modules. By default, this rule is already enabled.

**WANTHSREGEX** Apache requires a POSIX-compliant regular expression library. Henry Spencer’s excellent *Regex* package is included with Apache and is used automatically when the underlying operating system has no equivalent library. By default, this rule is enabled unless it is overruled by operating system specifics.

**EAPI (mod\_ssl only)** Enables EAPI, which provides a generic, low-level, function-calling mechanism, a generic data structure context mechanism; and shared memory support.

**SSL\_COMPAT (mod\_ssl only)** Enables `mod_ssl` to be built with backward-compatible code for Apache-SSL 1.x, `mod_ssl` 2.0.x, Sioux 1.x, and Stronghold 2.x. By default, it is already enabled.

**SSL\_SDBM (mod\_ssl only)** Controls whether the built-in SDBM library should be used for `mod_ssl` instead of a custom-defined or vendor-supplied DBM library. The default is to use a vendor NDBM library.

**SSL\_EXPERIMENTAL (mod\_ssl only)** Can be used to enable experimental code inside `mod_ssl`. These new features typically need more testing before they can be considered stable.

**SSL\_VENDOR (mod\_ssl only)** Can be used to enable code inside `mod_ssl` that product vendors can use to extend `mod_ssl` itself via EAPI hooks without patching the source.

■ **--disable-rule=Name** Disable a Configuration Rule

Example: `--disable-rule=WANTHSREGEX`

Disables a rule.

If you have a broken vendor regex library (for instance, if you observe core dumps on `RewriteRule` directives), use `--enable-rule=WANTHSREGEX`.

If you have a broken vendor NDBM library (for instance, if you observe core dumps on HTTPS requests), use `--enable-rule=SSL_SDBM`.

- **--add-module=*Name*** Import a Third-Party Module

Example: `--add-module=/tmp/mod.foo.c`

Imports and activates a third-party module *File* into the Apache source tree under `src/modules/extra/`. See Section 3.3.2 for more details.

- **--activate-module=*File*** Activate Third-Party Module

Example: `--activate-module=src/modules/extra/mod.foo.o`

Activates a manually imported third-party module *File*, which must stay under `src/modules/`. See Section 3.3.2 for more details.

- **--permute-module=*Name1*:*Name2*** Permute Module Order

Example: `--permute-module=rewrite:alias`

An expert option that can be used to permute the order of modules. See Section 3.3.3 for more details.

- **--enable-module=*Name*** Enable a Module for Building

Example: `--enable-module=rewrite`

Enables a module “*mod\_Name*” for building. The following modules are available (a “*\**” indicates that it is enabled by default):

<code>http_core *</code>	<code>mod_dir *</code>	<code>mod_asis *</code>
<code>mod_so</code>	<code>mod_actions *</code>	<code>mod_autoindex *</code>
<code>mod_alias *</code>	<code>mod_negotiation *</code>	<code>mod_status *</code>
<code>mod_rewrite</code>	<code>mod_env *</code>	<code>mod_info</code>
<code>mod_userdir *</code>	<code>mod_setenvif *</code>	<code>mod_log_config *</code>
<code>mod_imap *</code>	<code>mod_unique_id</code>	<code>mod_log_agent</code>
<code>mod_speling</code>	<code>mod_cgi *</code>	<code>mod_log_referer</code>
<code>mod_access *</code>	<code>mod_include *</code>	<code>mod_usertrack</code>
<code>mod_auth *</code>	<code>mod_mime *</code>	<code>mod_mmap_static</code>
<code>mod_auth_anon</code>	<code>mod_mime_magic</code>	<code>mod_example</code>
<code>mod_auth_dbm</code>	<code>mod_expires</code>	<code>mod_proxy</code>
<code>mod_auth_db</code>	<code>mod_headers</code>	<code>mod_perl</code>
<code>mod_digest</code>	<code>mod_cern_meta</code>	<code>mod_ssl</code>

Two special variants of *Name* exist: “*all*” enables all existing modules and “*most*” enables only those modules known to be usable on all platforms without problems.

- **--disable-module=*Name*** Disable a Module for Building

Example: `--disable-module=alias`

Disables an enabled (by default or manually) module from building.

- **--enable-shared=*Name*** Enable a Module for DSO

Example: `--enable-shared=rewrite`

Use the powerful `--enable-module` and `--disable-module` options to assemble your individual Apache functionality.

Enables a module “*mod\_Name*” for building as a DSO. In addition to the standard module names, two special variants of *Name* exist: “*max*” enables DSO for all modules except for the bootstrapping modules (`http_core` and `mod_so`), and “*remain*” first enables all still-disabled modules, then enables them for building as DSO.

- **--disable-shared=*Name*** Disable a Module for DSO

Example: `--disable-shared=rewrite`

Disables a module “*mod\_Name*” for building as a DSO.

- **--with-perl=*File*** Sets the Perl Interpreter

Example: `--with-perl=/usr/local/bin/perl`

Sets the path to the Perl interpreter executable to *File*. By default, APACI searches for “*perl*” and “*perl5*” in `$PATH` for the latest interpreter version. Use this option when more than one Perl interpreter is installed on your system or you want to use a Perl interpreter found in a nonstandard file system location.

- **--without-support** Build without Support Tools

Example: `--without-support`

Forces APACI to not build the support tools under `src/support/`. By default, these tools are built. Use this option when these tools are unnecessary cause portability problems.

- **--without-confadjust** No Configuration Adjustments

Example: `--without-confadjust`

Forces APACI to not adjust the configuration files on installation. By default, APACI recognizes, for instance, that when you build as non-root (`UID ≠ 0`), it might be reasonable to pre-configure Apache for port 8080 instead of 80 (because non-root users cannot run Apache on port 80). Sometimes these adjustments are confusing, especially for vendor package maintainers. In such a case, you can disable the adjustments with this option.

- **--without-execstrip** No Executable Stripping

Example: `--without-execstrip`

Forces APACI to not “strip” (remove debugging symbols) the executables when installing them. This option can be either useful for debugging purposes or required on esoteric platforms where the DSO facility works only when the Apache executable is not “stripped.”

Vendor package maintainers should keep `--without-confadjust` in mind.

### 3.2.6 suEXEC Options

The following *options* for configuring the suEXEC facility are available on the APACI command line:

- **--enable-suexec** Enables suEXEC Facility  
 Example: `--enable-suexec`  
 Enables the suEXEC facility, which can be used to run CGI scripts under particular UIDs.
- **--suexec-caller=*Name*** suEXEC Caller UID  
 Example: `--suexec-caller=www`  
 Sets the user name of the suEXEC calling process to *Name* — that is, the UID under which Apache *runs* (when Apache is started as root, so that UID = 0, the *Name* can be a configured custom UID; see the `User` directive). The suEXEC then runs only CGI scripts when the calling process has this user name.
- **--suexec-docroot=*Dir*** suEXEC Document Root  
 Example: `--suexec-docroot=/usr/local/apache/share/htdocs`  
 Sets the path for the suEXEC document root to *Dir*.
- **--suexec-logfile=*File*** suEXEC Log File Path  
 Example: `--suexec-logfile=/usr/local/apache/var/suexec.log`  
 Sets the path for the dedicated suEXEC log file to *File*.
- **--suexec-userdir=*SubDir*** suEXEC User Home Subdirectory  
 Example: `--suexec-userdir=.public-html`  
 Sets *SubDir* as the subdirectory of the user's "homedirs," where CGI scripts must reside to be executed through suEXEC.
- **--suexec-uidmin=*UID*** suEXEC Minimum UID  
 Example: `--suexec-uidmin=1024`  
 Sets the minimum UNIX user ID to *UID*; the suEXEC facility can then switch to it.
- **--suexec-gidmin=*GID*** suEXEC Minimum GID  
 Example: `--suexec-gidmin=1024`  
 Sets the minimum UNIX group ID to *GID*; the suEXEC facility can then switch to it.
- **--suexec-safepath=*Path*** suEXEC Safe PATH Variable  
 Example: `--suexec-safepath=/bin:/usr/bin`  
 Enforces the colon-separated `$PATH` variable to *Path* for use under the suEXEC facility.

The suEXEC facility allows CGI scripts to be executed under the UID/GID of the script owner instead of the runtime UID/GID of the Apache server processes.

## 3.3 Configuration Special Topics

The final section of this chapter examines some special configuration issues on which we've touched only tangentially in previous discussions.

### 3.3.1 Shadow Source Trees

The `--shadow[=Dir]` option is very interesting. It can be used to build Apache inside a temporary location without copying the entire Apache source tree (15MB in size). This option is useful mainly in two situations. First, you can use it when you want to build Apache on a cluster of machines in parallel and want to avoid conflicts (the source then generally stays on an NFS-mounted file system). Second, when the Apache source tree resides on a read-only file system (typically a CD-ROM), you must ensure that the build process can write the object files. Both problems are efficiently resolved through shadow trees.

A shadow tree consists of a copy of all directories of the original tree, but with all files inside these directories being replaced by symbolic links to the original files. Such a tree can be created more quickly than a direct tree copy can, and it requires less disk space. You simply specify an additional `--shadow` option on the APACI command line, and Apache automatically builds inside this tree in the background.

Shadow trees may be employed in two ways:

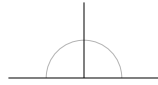
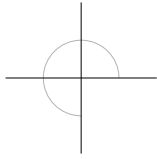
- You can specify only `--shadow`. In this case, the shadow tree is made only for the `src/` subdirectory of the Apache source tree and placed side-by-side to this directory. It is named `src.platform`, where *platform* is the platform identification string. Use this option when you want to build for multiple architectures in parallel.
- You can specify `--shadow=Dir`. In this case, the shadow tree is made for the entire Apache source tree and placed under *Dir*. Use this option when you want to build from a read-only media.

### 3.3.2 On-the-Fly Addition of Third-Party Modules

As you may have recognized in our example installation (Section 3.1 on page 37), third-party modules can be added to the Apache source tree in three ways:

- They can be automatically added and activated by a script. For instance, `mod_ssl` uses this approach.

The flexibility of Apache means that one can easily add third-party modules to extend Apache's functionality.



- They can be automatically added by a script but activated manually by the user. For instance, `mod_perl` uses this approach.
- They can be manually added and activated by the user. Most Apache modules provided by third parties use this approach.

This little inconsistency arises because larger modules have more requirements; to make the life of the user easier, these modules partly automate the steps. Don't be alarmed if the complex modules differ. The basic way for manually adding a third-party module in APACI involves three steps:

1. Obtain the module source. For small modules, it is typically a `mod_foo.c` source file. For larger modules, it may be a directory containing at least `Makefile.tmp1`, `mod_bar.c`, and a few additional source files (conventionally named `bar_xxx.c`).
2. Add the module source to the Apache source tree somewhere under `src/modules`. The location selected depends on the module. With a single `mod_foo.c`, you usually place the source under `src/modules/extra/` by using `--add-module=/path/to/foo/mod_foo.c`. For larger modules that require their own subdirectory under `src/modules/` (say, `src/modules/bar`), you must establish this directory manually by running `cp -rp /path/to/bar/ src/modules/bar/` and later activate it by using `--activate-module=src/modules/bar/libbar.a`.
3. Once the third-party modules are fully integrated into the source tree of Apache, you can treat them just like the distributed modules. In both cases, you enable the module for building via `--enable-module=foo` or `--enable-module=bar`. The same idea applies when building as a DSO: a simple `--enable-shared=foo` or `--enable-shared=bar` is all that is needed.

Third-party modules differ mainly in size: either they are single source modules or they are contained in their own directory.



### 3.3.3 Module Order and Permutations

You may have recognized the harmless-looking APACI option `--permutemodule=Name1:Name2`. We briefly mentioned that it can be used for changing the order of modules. To fully understand this option and its utility, more knowledge of Apache internals are required.

As explained in Chapter 2, the functionality of Apache is implemented by modules. An API dispatches the HTTP request processing. During this dispatching a fixed module order is used that is derived from the order employed when building the modules. Actually, it mirrors the order of the `AddModule/SharedModule` lines in the file `src/Configuration.apaci`, which APACI generates from `src/Configuration.tmp1`. When a module

Use `--permutemodule` to change the module order at installation time and the execution order at runtime.

comes later in this file, it is dispatched earlier in the processing. For instance, `mod_rewrite` comes after `mod_alias` in this file, so `mod_rewrite` gets control for each API step before `mod_alias`. As a result, `mod_rewrite` can manipulate URLs before `mod_alias` can, but it cannot override results of `mod_alias`.

For all distributed modules, the order is pre-configured in a reasonable way. Nevertheless, sometimes you may want to change the order of one or more modules. For instance, to give `mod_alias` higher priority over `mod_rewrite`, you would use the following:

```
$ ./configure ... --permute-module=alias:rewrite ...
```

Now `mod_rewrite` can post-process URLs that were manipulated by `mod_alias`. On the other hand, when you add a third-party module, it is always appended to the end of `src/Configuration.apaci`; hence, it gets the highest priority by default. This order often isn't reasonable. For instance, when you have added another URL manipulation module (say, `mod_foo`), it might be reasonable to ensure that it operates after `mod_rewrite` and `mod_alias`. This goal can be achieved by using the following APACI command line:

```
$ ./configure ... --add-module=/path/to/mod_foo.c --permute-module=foo:BEGIN ...
```

This command moves `mod_foo` to the beginning of the module list and gives it the lowest priority. More complex module order adjustments can be achieved by combining multiple `--permute-module` options.