
Adding Common Modules

There are a number of extremely popular modules for the Apache Web server that are not included in the basic distribution. Most of these are separate because of licensing or support reasons; some are not distributed by the Apache Software Foundation because of a decision by the Apache developers; and some are integral parts of other projects. For instance, *mod_ssl* for Apache 1.3 is developed and maintained separately not only because of the U.S. export control laws (which were more restrictive when the package was originally developed), but because it requires changes to the core software that the Apache developers chose not to integrate.

This chapter provides recipes for installing some of the most popular of these third-party modules; when available, there are separate recipes for installation on Unixish systems and on Windows.

The most comprehensive list of third-party modules can be found in the Apache Module Registry at <http://modules.apache.org/>. Some modules are so popular—or complex—that they have entire sites devoted to them, as do the ones listed in this chapter.

Although hundreds of third-party modules are available, many module developers are only concerned with their single module. This means that there are potentially as many different sets of installation instructions as there are modules. The first recipe in this chapter describes an installation process that should work with many Apache 1.3 modules, but you should check with the individual packages' instructions to see if they have a different or more detailed process.

Many of the modules are available from organizations that prepackage or distribute Apache software, such as in the form of an RPM from Mandrake or Red Hat, but such prebuilt module packages include the assumptions of the packager. In other words, if you build the server from source and use custom locations for the files, don't be surprised if the installation of a packaged module fails.

All of the modules described in this chapter are supported with Apache 1.3 on Unixish systems. Status of support with Apache 2.0 on Windows is shown in Table 2-1.

Table 2-1. Module support status

Module name	Windows	Support on Apache 2.0
<i>mod_dav</i>	Yes	Included; no installation necessary
<i>mod_perl</i>	Yes	Yes
<i>mod_php</i>	Yes	Yes
<i>mod_ssl</i>	No	Included; no installation necessary

2.1 Installing a Generic Third-Party Module

Problem

You have downloaded a third-party module that isn't listed in this chapter, and you want to install it.

Solution

Move to the directory where the module's source file was unpacked, and then:

```
% /path/to/apache/bin/apxs -cia module.c
```

Discussion

In the case of a third-party module that consists of a single *.c* file, there is a good chance that it can be built and installed using the Solution. Modules that involve multiple source files should provide their own installation instructions.

The *-cia* options mean to compile, install, and activate. The first is pretty straightforward; install means put the *.so* file in the place Apache expects to find it, and activate means to add the module to the *httpd.conf* file.

See Also

- The *apxs* manpage, typically *ServerRoot/man/man8/apxs.8*

2.2 Installing *mod_dav* on a Unixish System

Problem

You want to add or enable WebDAV capabilities to your server. WebDAV permits specific documents to be reliably and securely manipulated by remote users without the need for FTP, to perform such tasks as adding, deleting, or updating files.

Solution

If you're using Apache 2.0 or later, *mod_dav* is automatically available, although you may need to enable it at compile time with `--enable-dav`.

If you are using Apache 1.3, download and unpack the *mod_dav* source package from http://webdav.org/mod_dav/, and then:

```
% cd mod_dav-1.0.3-1.3.6
% ./configure --with-apxs=/usr/local/apache/bin/apxs
% make
# make install
```

Restart the server, and be sure to read Recipe 6.18.

Discussion

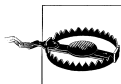
mod_dav is an encapsulated and well-behaved module that is easily built and added to an existing server. To test that it has been properly installed, you need to enable some location on the server for WebDAV management and verify access to that location with some WebDAV-capable tool. We recommend *cadaver*, which is an open source command-line WebDAV tool. (The URL for the *cadaver* tool is found at the end of this recipe.)

To enable your server for WebDAV operations, you need to add at least two directives to your *httpd.conf* file. The first identifies the location of the locking database used by *mod_dav* to keep WebDAV operations from interfering with each other; it needs to be in a directory that is writable by the server. For example:

```
# cd /usr/local/apache
# mkdir var
# chgrp nobody var
# chmod g+w var
```

Now add the following line to your *httpd.conf* file, outside any containers:

```
<IfModule mod_dav.c>
    DAVLockDB var/DAVLock
</IfModule>
```



The *DAVLockDB* location *must not* be on an NFS-mounted filesystem, because NFS doesn't support the sort of locking *mod_dav* requires. Putting the lock database on an NFS filesystem may result in unpredictable results.

Next, create a temporary directory for testing WebDAV functionality:

```
# cd /usr/local/apache
# mkdir htdocs/dav-test
# chgrp nobody htdocs/dav-test
# chmod g+w htdocs/dav-test
```

Add a stanza to your *httpd.conf* file that will enable this directory for WebDAV operations:

```
<Directory "/usr/local/apache/htdocs/dav-test">
    DAV On
</Directory>
```

Now restart your server. It should be ready to handle WebDAV operations directed to the */dav-test* local URI. To test it with the *cadaver* tool, try the following commands; your output should look very similar to that shown:

```
% cd /tmp
% echo "Plain text" > dav-test.txt
% cadaver
dav: !> open http://localhost/dav-test
Looking up hostname... Connecting to server... connected.
dav:/dav-test/> put dav-test.txt
Uploading dav-test.txt to '/dav-test/dav-test.txt': (reconnecting...done)
Progress: [= == == == == == == == == == == == == == ==>] 100.0% of 11 bytes succeeded.
dav:/dav-test/> propset dav-test.txt MyProp 1023
Setting property on 'dav-test.txt': (reconnecting...done) succeeded.
dav:/dav-test/> propget dav-test.txt MyProp
Fetching properties for 'dav-test.txt':
Value of MyProp is: 1023
dav:/dav-test/> propdel dav-test.txt MyProp
Deleting property on 'dav-test.txt': succeeded.
dav:/dav-test/> close
Connection to 'localhost' closed.
dav: !> exit
% rm dav-test.txt
```

Properties are attributes of a WebDAV resource. Some are managed by the system, such as the resource's size, but others can be arbitrary and added, changed, and removed by the user.

Once you have verified that *mod_dav* is working correctly, remove the *htdocs/dav-test* directory, and the corresponding *<Directory>* stanza in your *httpd.conf* file, and follow the guidelines in Recipe 6.18.

See Also

- Recipe 6.18
- http://webdav.org/mod_dav/
- <http://webdav.org/cadaver/>

2.3 Installing mod_dav on Windows

Problem

You want to enable WebDAV capabilities on your existing Apache 1.3 server with *mod_dav*.

Solution

Apache 2.0 includes *mod_dav* as a standard module, so you do not need to download and build it.

Download and unpack the *mod_dav* Windows package from http://webdav.org/mod_dav/win32/. Verify that your Apache installation already has the *xmlparse.dll* and *xmltok.dll* files in the *ServerRoot* directory; if they aren't there, check through the Apache directories to locate and copy them to the *ServerRoot*. *mod_dav* requires the *Expat* package, which is included with versions of the Apache Web server after 1.3.9; these files hook into *Expat*, which *mod_dav* will use.

Put the *mod_dav* DLL file into the directory where Apache keeps its modules:

```
C:\>cd mod_dav-1.0.3-dev
C:\mod_dav-1.0.3-dev>copy mod_dav.dll C:\Apache\modules
C:\mod_dav-1.0.3-dev>cd \Apache
```

Add the following line to your *httpd.conf* file:

```
LoadModule dav_module modules/mod_dav.dll
```

You may also need to add an *AddModule* line if your *httpd.conf* file includes a *ClearModuleList* directive and re-adds the other modules. Alternatively, you can insert the *LoadModule* for *mod_dav* after the *ClearModuleList* directive.

Discussion

mod_dav is an encapsulated and well-behaved module that is easily built and added to an existing server. To test that it has been properly installed, you need to enable some location on the server for WebDAV management and verify access to that location with some WebDAV-capable tool, or browse to it in *Windows Explorer*, which knows how to access WebDAV locations (as of Windows 2000), or access it from a different system where *cadaver* or another WebDAV tool is available.

To enable your server for WebDAV operations, you need to add at least two directives to your *ServerRoot/conf/httpd.conf* file. The first identifies the location of the locking database used by *mod_dav* to keep WebDAV operations from interfering with each other; it needs to be in a directory that is writable by the server. For example:

```
C:\Apache-1.3>mkdir var
```

Now add the following lines to your *httpd.conf* file to enable WebDAV:

```
<IfModule mod_dav.c>
    DAVLockDB "C:/Apache-1.3/var/dav-lock"
</IfModule>
```

Create a temporary directory for testing *mod_dav*'s ability to function:

```
C:\Apache-1.3>mkdir htdocs\dav-test
```

Modify the *<IfModule>* container to enable WebDAV operations for this test directory:

```
<IfModule mod_dav.c>
    DAVLockDB "C:/Apache-1.3/var/dav-lock"
    <Directory "C:/Apache-1.3/htdocs/dav-test">
        DAV On
    </Directory>
</IfModule>
```

Now restart your server and try accessing the */dav-test* location with a WebDAV client. If you're using *cadaver* from another system, see Recipe 2.2 for detailed instructions. If you want to use *Windows Explorer* to test *mod_dav*, read the following section.

Using Windows Explorer to test mod_dav

After enabling the *htdocs\dav-test* directory for WebDAV operations and restarting your server, start up *Windows Explorer*. Follow the steps below to access the directory using WebDAV. This can be done on the local system or on another Windows system that can access your server system.

1. Click on Network Places.
2. In the righthand pane of the *Windows Explorer* window, you should see an item named Add Network Place. Double-click on this item.
3. When prompted for a location, enter:

```
http://127.0.0.1/dav-test/
```

If you are executing these steps on a different system, replace the 127.0.0.1 with the correct name of the server on which you installed *mod_dav*.

4. After clicking on Next, give this location any name you like or keep the default.
5. After completing the dialog, *Windows Explorer* should open a new window with the name you selected in the previous step. The window should be empty, which makes sense since the directory is.
6. In the main *Windows Explorer* window, navigate to a directory (any directory) with files in it.
7. Ctrl-drag a file (any file) from the main *Windows Explorer* window to the window that was opened by step 5.
8. Windows should briefly display a progress dialog window, and then the file should appear in the destination window.

Congratulations! The file was uploaded to your Web server using WebDAV.

After your testing is complete, don't forget to remove the *htdocs\dav-test* directory and the `<Directory "C:/Apache-1.3/htdocs/dav-test">` stanza in your configuration file, or else anyone can upload files to your server.

See Also

- Recipe 6.18
- http://webdav.org/mod_dav/

2.4 Installing *mod_perl* on a Unixish System

Problem

You want to install the *mod_perl* scripting module to allow better Perl script performance and easy integration with the Web server.

Solution

For Apache 1.3, download and unpack the *mod_perl* 1.0 source package from <http://perl.apache.org/>. Then use the following command:

```
% perl Makefile.PL \
>   USE_APXS=1 \
>   WITH_APXS=/usr/local/apache/bin/apxs \
>   EVERYTHING=1 \
>   PERL_USELARGEFILES=0
% make
% make install
```

Restart your server.

For Apache 2.0 and later, the process is similar. Download and unpack the *mod_perl* 2.0 source package, then use the following command:

```
% perl Makefile.PL MP_APXS=/usr/local/apach2/bin/apxs
```

Discussion

mod_perl is quite a complex module, and there are several different ways to add it to your server. This recipe is the fastest and lowest-impact one; if it doesn't suit your needs, check the various *README.** files in the package directory after unpacking. Because its primary language is Perl rather than C, the installation instructions are significantly different from those for most other modules.

Once you have restarted your server successfully, *mod_perl* should be available and configured as part of it. You can test it by making some changes to the *httpd.conf* file, adding a few scripts, and seeing whether the server processes them correctly. Here is a sample set of steps to test *mod_perl*'s operation.

1. Create a directory where your *mod_perl* scripts can live:

```
# cd ServerRoot
# mkdir lib lib/perl lib/perl/apache
```

2. Create a file named *startup.pl* in your server's *conf/* directory that will give *mod_perl* some startup instructions:


```
#!/usr/bin/perl
BEGIN {
    use Apache ( );
    use lib Apache->server_root_relative('lib/perl');
}
use Apache::Registry ( );
use Apache::Constants ( );
use CGI qw(-compile :all);
use CGI::Carp ( );
1;
```

3. Next, create the *lib/perl/apache/HelloWorld.pm* file that will be used for our test:

```
package Apache::HelloWorld;
use strict;
use Apache::Constants qw(:common);
sub handler {
    my $r = shift;
    $r->content_type('text/plain; charset=ISO-8859-1');
    $r->send_http_header;
    $r->print("Hello, world! Love, mod_perl.\n");
    return OK;
}
1;
```

4. Next, edit the server's configuration file to add the directives that will enable *mod_perl* to locate all the pieces it needs, and tell it when to invoke the test script. Add the following lines to the *httpd.conf* file:

```
<IfModule mod_perl.c>
    PerlRequire conf/startup.pl
    <Location /mod_perl/howdy>
        SetHandler perl-script
        PerlHandler Apache::HelloWorld
    </Location>
</IfModule>
```

5. Now restart your server and request the script using *http://localhost/mod_perl/howdy*. 

If your configuration is valid, the response should be a page containing simply the words, "Hello, world! Love, mod_perl."

See Also

- <http://perl.apache.org/>

- *Writing Apache Modules with Perl and C* by Doug MacEachern and Lincoln Stein (O'Reilly)
- *mod_perl Developer's Cookbook* by Geoffrey Young, Paul Lindner, and Randy Kobes (Sams)

2.5 Installing *mod_php* on a Unixish System

Problem

You want to add the *mod_php* scripting module to your existing Apache Web server.

Solution

Download the *mod_php* package source from the Web site at <http://php.net/> (follow the links for downloading) and unpack it. Then:

```
% cd php-5.2.3
% ./configure \
> --with-apxs2=/usr/local/apache/bin/apxs
% make
# make install
```

Restart the server.

Discussion

To test that your installation was successful, create a file named *info.php* in your server's *DocumentRoot*; the file should contain the single line:

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

Add the following lines to your server's *httpd.conf* file:

```
<IfModule mod_php4.c>
    AddHandler application/x-httpd-php .php
</IfModule>
```

After restarting your server, try fetching the document *info.php* using a browser. You should see a detailed description of the PHP options that are active. If you do, indicating a successful installation, remove the *info.php* file.

There are numerous additional options and extensions available for PHP; the recipe given here is only for the most basic installation.

See Also

- Recipe 8.16
- Recipe 8.17
- <http://php.net/>

2.6 Installing *mod_php* on Windows

Problem

You want to add the *mod_php* scripting module to your existing Apache server on Windows.

Solution

This recipe needs to be described largely in terms of actions rather than explicit commands to be issued.

1. Download the PHP Windows binary *.zip* file with API extensions (not the *.exe* file) from <http://php.net/>.
2. Unpack the *.zip* file into a directory where you can keep its contents indefinitely (such as *C:\PHP4*). If you use *WinZip*, be sure to select the Use folder names checkbox to preserve the directory structure inside the *.zip* file.
3. Copy the *PHP4SAPI\php4apache.dll* file to the *\modules* directory under your Apache installation's *ServerRoot*.
4. In a command-prompt window, change to the *PHP4* directory where you unpacked the *.zip* file, and type:

```
... \PHP4>copy php.ini-dist %SYSTEMROOT%\php.ini
... \PHP4>copy php4ts.dll %SYSTEMROOT%
```

(If installing on Windows 95 or Windows 98, use *%WINDOWS%* instead of *%SYSTEMROOT%*.)

5. Edit the *%SYSTEMROOT%\php.ini* file, locate the line that starts with *extensions_dir*, and change the value to point to the *PHP4\extensions* directory. For instance, if you unpacked the *.zip* file into *C:\PHP4*, this line should look like:

```
extensions_dir = C:\PHP4\extensions
```

6. Edit the *conf\httpd.conf* file under the Apache *ServerRoot* and add the following lines near the other *LoadModule* lines:

```
LoadModule php4_module modules/php4apache.dll
```

Add the following lines in some scope where they will apply to your *.php* files:

```
<IfModule mod_php4.c>
    AddType application/x-httpd-php .php
</IfModule>
```

7. Restart the Apache server, and the PHP module should be active.

Discussion

The PHP module installation on Windows requires a lot of nitpicky manual steps. To test that your installation was successful, create a file named *info.php* in your server's *DocumentRoot*; the file should contain the single line:

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

After restarting your server, try fetching the document *info.php* from it using a browser. You should see a detailed description of the PHP options that are active.

There are numerous additional options and extensions available for PHP; the recipe given here is only the most basic installation. See the *install.txt* file in the *PHP4* directory and the documentation on the Web site for more details.

See Also

- <http://php.net/>

2.7 Installing *mod_ssl*

Problem

You want to add SSL support to your Apache server with the *mod_ssl* secure HTTP module.

Solution

Windows

There is a discussion of installing SSL on Windows in Recipe 7.2, but the short form is, you should get XAMPP from ApacheFriends.org, unless you are very experienced with building source code on the Microsoft Windows platform.

Apache 2.0

mod_ssl is included with 2.0, although it is not automatically compiled nor installed when you build from source. You need to include the `--enable-ssl` option on your *./configure* line, and enable it with *LoadModule* and *AddModule* directives.

Apache 1.3

To install *mod_ssl* on a Unixish system, download the tarball package from the <http://www.modssl.org/> Web site and unpack it. Then:

```
% cd mod_ssl-2.8.14-1.3.27
% ./configure \
> --with-apache=../apache_1.3.27 \
> --with-ssl=SYSTEM \
> --prefix=/usr/local/apache
% cd ../apache_1.3.27
% make
% make certificate
```

Discussion

The *mod_ssl* package requires source-level changes to the base Apache code, and so the version of the *mod_ssl* package you install must match the version of the Apache distribution you have. If your Apache installation doesn't include the source (such as if you installed a binary-only RPM or other vendor distribution), you won't be able to add *mod_ssl* to it.

In addition to the Apache source, *mod_ssl* requires that you have Perl and the OpenSSL libraries installed. The `--with-ssl` option on the build configuration statement indicates where this is located; if it is in a vendor distributed directory, the special keyword `SYSTEM` tells the build to look for it, and you don't have to find it yourself.

Unlike most other Apache modules, when adding *mod_ssl* you run the *./configure* script that's in *mod_ssl*'s directory rather than the one in the Apache source directory; the module's script makes changes to Apache's and then invokes it directly.

This recipe is the bare basics; there are many optional components and features that *mod_ssl* allows you to specify at configuration time. For more information, consult the *README* and *INSTALL* files in the *mod_ssl* source directory, or the *mod_ssl* Web site at <http://www.modssl.org/>.

See Also

- Recipe 7.3
- <http://www.modssl.org/>

2.8 Finding Modules Using Modules.Apache.Org

Problem

You're looking for Apache modules with a particular functionality, or by name, and you've heard about the Apache Module Registry.

Solution

Visit <http://modules.apache.org/> and search for keywords related to the functionality you want, or portions of the module name.

Discussion

The Apache modules registry is an unofficial site at which module authors can voluntarily register their work for easy location.



By no means are all third-party modules registered on this site; many are on SourceForge or on their authors' home systems. If you don't find what you're looking for at <http://modules.apache.org/>, try SourceForge (<http://sourceforge.net/>), FreshMeat (<http://freshmeat.net/>), or just search the Web with Google or the search engine of your choice.

See Also

- <http://SourceForge.Net/>
- <http://FreshMeat.Net/>

2.9 Installing *mod_security*

Problem

You want to install the *mod_security* module to take advantage of its simple and powerful filtering mechanisms.

Solution

- Download *mod_security* and the core rules from <http://modsecurity.org/download/>.



After downloading, you should verify the PGP signature to make sure the file hasn't been altered. See the *mod_security* Web site for details.

- Unpack the kit (not the rules) into a working directory:

```
% cd /usr/local/build
% tar xzf /usr/local/kits/modsecurity-apache_2.1.1
```

- Move into the unpacked directory, and build the package using the supplied *Make* file. Specify the value of your *ServerRoot* on the *make* command line:

```
% cd /usr/local/build/modsecurity-apache_2.1.1/apache2
% make top_dir=/usr/local/apache2
# make top_dir=/usr/local/apache2 install
```



Unlike many other third-party modules, *mod_security* needs to be built using its own mechanism rather than a simple invocation of Apache's *apxs* tool.

- Unpack the core rules into a subdirectory under your *ServerRoot*:

```
# cd /usr/local/apache2/conf
# mkdir mod_security
```

```
# cd mod_security
# tar xzf /tmp/modsecurity-core-rules_2.1-1.4.tar.gz
```

- Edit your *httpd.conf* file to add the following lines in the appropriate places:

```
LoadModule security_module modules/mod_security2.so

Include conf/mod_security/*.conf
```

- Restart your server.

Discussion

The *Makefile* included with the *mod_security* package will do the building of the module and put it in the right place, but activating it in your server is your responsibility. Recent versions of the package include a set of core rules for handling things like blog spam and common attacks, and the rules are also available as a separate tarball (which may or may not be updated more frequently than the ones bundled with the software).

The current version of *mod_security* only supports version 2 of the Apache Web server. There is an older version that supports the 1.3 versions, but it is unlikely to be maintained for long.

See Also

- The *mod_security* Web site at <http://modsecurity.org/>.

2.10 Why Won't This Module Work?

Problem

You are trying to install a third-party module, but the Apache Web server refuses to recognise it.

Solution

Consult the sources for the module, or its documentation, or ask the author, in order to determine which version of Apache the package supports.

Discussion

As significant changes are made to the Apache Web server, sometimes compatibility suffers as the API is changed. Although efforts are made to keep this sort of thing to a minimum, sometimes it is unavoidable.

To keep an incompatible module from being loaded and crashing the Web server when used, both modules and the server have a built-in 'magic' number which is recorded when they're built, and that relates to the version of the API. When the server tries to load a module DSO, it compares the module's magic number with the server's own, and if they aren't compatible the server refuses to load it.

The development team tries to keep the magic number compatibility within major version numbers, but not across them. That is, a module built for Apache 1.3 *should* work with almost any 1.3 version of the server built after the module was, but it definitely won't work with a 2.0 server. Contrariwise, a 2.0 module won't work with a 1.3 server under any circumstances.

See Also

- The Apache modules registry at <http://modules.apache.org/>