1 mod_perl Installation

1.1 Description

An in-depth explanation of the mod_perl installation process, from the basic installation (in 10 steps), to a more complex one (with all the possible options you might want to use, including DSO build). It includes troubleshooting tips too.

First of all:

```
Apache 2.0 doesn't work with mod_perl 1.0.
Apache 1.0 doesn't work with mod_perl 2.0.
```

1.2 A Summary of a Basic mod_perl Installation

The following 10 commands summarize the execution steps required to build and install a basic mod_perl enabled Apache server on almost any standard flavor of Unix OS.

```
% cd /usr/src
% lwp-download http://www.apache.org/dist/httpd/apache_1.3.xx.tar.gz
% lwp-download http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
% tar xzvf apache_1.3.xx.tar.gz
% tar xzvf mod_perl-1.xx.tar.gz
% cd mod_perl-1.xx
% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
% make && make test && make install
% cd ../apache_1.3.xx
% make install
```

Of course you should replace *1.xx* and *1.3.x* with the real version numbers of mod_perl and Apache.

All that's left is to add a few configuration lines to `httpd.conf`, the Apache configuration file, start the server and enjoy mod_perl.

If you have stumbled upon a problem at any of the above steps, don't despair, the next sections will explain in detail each and every step.

Of course there is a way of installing mod_perl in only a couple of minutes if you are using a Linux distribution that uses RPM packages:

```
% rpm -i apache-1.3.xx.rpm
% rpm -i mod_perl-1.xx.rpm
```

or apt system:

```
% apt-get install libapache-mod-perl
```

These should set up both Apache and mod_perl correctly for your system. Using a packaged distribution can make installing and reinstalling a lot quicker and easier. (Note that the filenames will vary, and *xx* will differ.)

Since mod_perl can be configured in many different ways (features can be enabled or disabled, directories can be modified, etc.) it's preferable to use a manual installation, as a prepackaged version might not suit your needs. Manual installation will allow you to make the fine tuning for the best performance as well.

In this chapter we will talk extensively about the prepackaged versions, and ways to prepare your own packages for reuse on many machines. Win32 users should consult the Win32 documentation for details on that platform.

1.3 The Gory Details

We saw that the basic mod_perl installation is quite simple and takes about 10 commands. You can copy and paste them from these pages. The parameter `EVERYTHING=1` selects a lot of options, but sometimes you will need different ones. You may need to pass only specific parameters, to bundle other components with mod_perl etc. You may want to build mod_perl as a loadable object instead of compiling it into Apache, so that it can be upgraded without rebuilding Apache itself.

To accomplish this you will need to understand various techniques for mod_perl configuration and building. You need to know what configuration parameters are available to you and when and how to use them.

As with Perl, with mod_perl simple things are simple. But when you need to accomplish more complicated tasks you may have to invest some time to gain a deeper understanding of the process. In this chapter I will take the following route. I'll start with a detailed explanation of the four stages of the mod_perl installation process, then continue with the different paths each installation might take according to your goal, followed by a few copy-and-paste real world installation scenarios. Towards the end of the chapter I will show you various approaches that make the installations easier, automating most of the steps. Finally I'll cover some of the general issues that can cause new users to stumble while installing mod_perl.

We can clearly separate the installation process into the following stages:

- **Source Configuration,**
- **Building,**
- **Testing and**
- **Installation.**

1.3.1 Source Configuration (perl Makefile.PL ...)

Before building and installing mod_perl you have to configure it. You configure mod_perl just like any other Perl module:

```
  % perl Makefile.PL [parameters]
```

In this section we will go through most of the parameters mod_perl can accept and explain each one of them.

First let's see what configuration mechanisms we have available. Basically they all define a special set of parameters to be passed to `perl Makefile.PL`. Depending on the chosen configuration, the final product might be a stand-alone httpd binary or a loadable object.

The source configuration mechanism in Apache 1.3 provides four major features (which of course are available to mod_perl):

- **Per-module configuration scripts (ConfigStart/End)**

  This is a mechanism modules can use to link themselves into the configuration process. It is useful for automatically adjusting the configuration and build parameters from the modules sources. It is triggered by `ConfigStart/ConfigEnd` sections inside *modulename.module* files (e.g. *libperl.module*).

- **Apache Autoconf-style Interface (APACI)**

  This is the new top-level `configure` script from Apache 1.3 which provides a GNU Autoconf-style interface. It is useful for configuring the source tree without manually editing any *src/Configuration* files. Any parameterization can be done via command line options to the `configure` script. Internally this is just a nifty wrapper to the old `src/Configure` script.

  Since Apache 1.3 this is the way to install mod_perl as cleanly as possible. Currently this is a pure Unix-based solution because at present the complete Apache 1.3 source configuration mechanism is only available under Unix. It doesn't work on the Win32 platform for example.

- **Dynamic Shared Object (DSO) support**

  Besides Windows NT support this is one of most interesting features in Apache 1.3. Its a way to build Apache modules as so-called *dynamic shared objects* (usually named *modulename.so*) which can be loaded via the `LoadModule` directive in Apache's *httpd.conf* file. The benefit is that the modules are part of the `httpd` executable only on demand, i.e. only when the user wants a module it is loaded into the address space of the `httpd` executable. This is interesting for instance in relation to memory consumption and upgrading.

  The DSO mechanism is provided by Apache's `mod_so` module which needs to be compiled into the `httpd` binary. This is done automatically when DSO is enabled for module `mod_foo` via:

  ```
  ./configure --enable-module=foo
  ```

  or by explicitly adding `mod_so` via:

  ```
  ./configure --enable-module=so
  ```

- **APache eXtenSion (APXS) support tool**

  This is a new support tool from Apache 1.3 which can be used to build an Apache module as a DSO even **outside** the Apache source-tree. One can say `APXS` is for Apache what `MakeMaker` and `XS` are for Perl. It knows the platform dependent build parameters for making DSO files and provides an easy way to run the build commands with them.

  (`MakeMaker` allows an easy automatic configuration, build, testing and installation of the Perl modules, and `XS` allows to call functions implemented in C/C++ from Perl code.)

Taking these four features together provides a way to integrate mod_perl into Apache in a very clean and smooth way. *No patching* of the Apache source tree is needed in the standard situation and in the APXS situation not even the Apache source tree is needed.

To benefit from the above features a new hybrid build environment was created for the Apache side of mod_perl. The Apache-side consists of the mod_perl C source files which have to be compiled into the `httpd` program. They are usually copied to the subdirectory *src/modules/perl/* in the Apache source tree. To integrate this subtree into the Apache build process a lot of adjustments were done by mod_perl's *Makefile.PL* in the past. And additionally the `Makefile.PL` controlled the Apache build process.

This approach is problematic in several ways. It is very restrictive and not very clean because it assumes that mod_perl is the only third-party module which has to be integrated into Apache.

The new approach described below avoids these problems. It prepares only the *src/modules/perl/* subtree inside the Apache source tree *without* adjusting or editing anything else. This way, no conflicts can occur. Instead, mod_perl is activated later (when the Apache source tree is configured, via APACI calls) and then it configures itself.

We will return to each of the above configuration mechanisms when describing different installation passes, once the overview of the four building steps is completed.

1.3.1.1 Configuration parameters

The command `perl Makefile.PL`, which is also known as a *"configuration stage"*, accepts various parameters. In this section we will learn what they are, and when should they be used.

1.3.1.1.1 APACHE_SRC

If you specify neither the `DO_HTTPD` nor the `NO_HTTPD` parameters you will be asked the following question during the configuration stage:

```
  Configure mod_perl with ../apache_1.3.xx/src ?
```

`APACHE_SRC` should be used to define Apache's source tree directory. For example:

```
  APACHE_SRC=../apache_1.3.xx/src
```

Unless `APACHE_SRC` is specified, *Makefile.PL* makes an intelligent guess by looking at the directories at the same level as the mod_perl sources and suggests a directory with the highest version of Apache found there.

Answering *'y'* confirms either *Makefile.PL*'s guess about the location of the tree, or the directory you have specified with `APACHE_SRC`.

If you use `DO_HTTPD=1` or `NO_HTTPD`, the first Apache source tree found or the one you have defined will be used for the rest of the build process.

1.3.1.1.2 DO_HTTPD, NO_HTTPD, PREP_HTTPD

Unless any of `DO_HTTPD`, `NO_HTTPD` or `PREP_HTTPD` is used, you will be prompted by the following question:

```
Shall I build httpd in ../apache_1.3.xx/src for you?
```

Answering *'y'* will make sure an httpd binary will be built in *../apache_1.3.xx/src* when you run `make`.

To avoid this prompt when the answer is *Yes* specify the following argument:

```
DO_HTTPD=1
```

Note that if you set `DO_HTTPD=1`, but do not use `APACHE_SRC=../apache_1.3.xx/src` then the first apache source tree found will be used to configure and build against. Therefore it's highly advised to always use an explicit `APACHE_SRC` parameter, to avoid confusion.

`PREP_HTTPD=1` just means default 'n' to the latter prompt, meaning: *Do not build (make) httpd in the Apache source tree*. But it will still ask you about Apache's source location even if you have used the `APACHE_SRC` parameter. Providing the `APACHE_SRC` parameter will just eliminate the need for `perl Makefile.PL` to make a guess.

To avoid the two prompts:

```
Configure mod_perl with ../apache_1.3.xx/src ?
Shall I build httpd in ../apache_1.3.xx/src for you?
```

and avoid building httpd, use:

```
NO_HTTPD=1
```

If you choose not to build the binary you will have to do that manually. We will talk about it later. In any case you will need to run `make install` in the mod_perl source tree, so the Perl side of mod_perl will be installed. Note that, `make test` won't work until you have built the server.

1.3.1.1.3 Callback Hooks

A callback hook (abbrev. *callback*) is a reference to a subroutine. In Perl we create callbacks with the $callback = \&subroutine syntax, where in this example, `$callback` contains a reference to the subroutine called *"subroutine"*. Callbacks are used when we want some action (subroutine call) to occur when some event takes place. Since we don't know exactly when the event will take place we give the event handler a callback to the subroutine we want executed. The handler will call our subroutine at the right time.

By default, most of the callback hooks except for `PerlHandler,PerlChildInitHandler`, `PerlChildExitHandler`, `PerlConnectionApi`, and `PerlServerApi` are turned off. You may enable them by editing *src/modules/perl/Makefile*, or when running `perl Makefile.PL`.

The possible parameters for the appropriate hooks are:

```
PERL_POST_READ_REQUEST
PERL_TRANS
PERL_INIT
PERL_RESTART (experimental)

PERL_HEADER_PARSER
PERL_AUTHEN
PERL_AUTHZ
PERL_ACCESS
PERL_TYPE
PERL_FIXUP
PERL_LOG
PERL_CLEANUP
PERL_CHILD_INIT
PERL_CHILD_EXIT
PERL_DISPATCH

PERL_STACKED_HANDLERS
PERL_METHOD_HANDLERS
PERL_SECTIONS
PERL_SSI
```

As with any parameters that are either defined or not, use `PERL_hookname=1` to enable them (e.g. `PERL_AUTHEN=1`).

To enable all, but the last 4 callback hooks use:

```
ALL_HOOKS=1
```

1.3.1.1.4 EVERYTHING

To enable everything set:

```
EVERYTHING=1
```

1.3.1.1.5 PERL_TRACE

To enable debug tracing set: `PERL_TRACE=1`

1.3.1.1.6 APACHE_HEADER_INSTALL

By default, the Apache source headers files are installed into the *$Config{sitearch-exp}/auto/Apache/include* directory.

The reason for installing the header files is to make life simpler for module authors/users when building/installing a module that taps into some Apache C functions, e.g. `Embperl`, `Apache::Peek`, etc.

If you don't wish to install these files use:

```
APACHE_HEADER_INSTALL=0
```

1.3.1.1.7 PERL_STATIC_EXTS

Normally, if an extension is statically linked with Perl it is listed in `Config.pm`'s `$Config{static_exts}`, in which case mod_perl will also statically link this extension with httpd. However, if an extension is statically linked with Perl after it is installed, it is not listed in `Config.pm`. You may either edit `Config.pm` and add these extensions, or configure mod_perl like this:

```
 perl Makefile.PL "PERL_STATIC_EXTS=Something::Static Another::One"
```

1.3.1.1.8 APACI_ARGS

When you use the `USE_APACI=1` parameter, you can tell `Makefile.PL` to pass any arguments you want to the Apache `./configure` utility, e.g:

```
  % perl Makefile.PL USE_APACI=1 \
  APACI_ARGS='--sbindir=/usr/local/httpd_perl/sbin, \
        --sysconfdir=/usr/local/httpd_perl/etc, \
        --localstatedir=/usr/local/httpd_perl/var, \
        --runtimedir=/usr/local/httpd_perl/var/run, \
        --logfiledir=/usr/local/httpd_perl/var/logs, \
        --proxycachedir=/usr/local/httpd_perl/var/proxy'
```

Notice that **all** `APACI_ARGS` (above) must be passed as one long line if you work with `t?csh`!!! However it works correctly as shown above (breaking the long lines with '\') with `(ba)?sh`. If you use `t?csh` it does not work, since `t?csh` passes the `APACI_ARGS` arguments to `./configure` leaving the newlines untouched, but stripping the backslashes. This causes all the arguments except the first to be ignored by the configuration process.

1.3.1.1.9 APACHE_PREFIX

Alternatively to:

```
  APACI_ARGS='--prefix=/usr/local/httpd_perl'
```

from the previous section you can use the `APACHE_PREFIX` parameter. When `USE_APACI` is enabled, this attribute will specify the *--prefix* option just like the above setting does.

In addition when the `APACHE_PREFIX` option is used `make install` be executed in the Apache source directory, which makes these two equivalent:

```
% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1 \
  APACI_ARGS='--prefix=/usr/local/httpd_perl'
% make && make test && make install
% cd ../apache_1.3.xx
% make install

% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1 \
  APACHE_PREFIX=/usr/local/httpd_perl
% make && make test && make install
```

Now you can pick your favorite installation method.

1.3.1.2 Environment Variables

There are a few environment variables that influence the build/test process.

1.3.1.2.1 APACHE_USER and APACHE_GROUP

You can use the environment variables `APACHE_USER` and `APACHE_GROUP` to override the default `User` and `Group` settings in the *httpd.conf* used for 'make test' stage. (Introduced in mod_perl v1.23.)

1.3.1.3 Reusing Configuration Parameters

When you have to upgrade the server, it's quite hard to remember what parameters were used in a mod_perl build. So it's better to save them in a file. For example if you create a file at *~/.mod_perl_build_options*, with contents:

```
APACHE_SRC=../apache_1.3.xx/src DO_HTTPD=1 USE_APACI=1 \
EVERYTHING=1
```

You can build the server with the following command:

```
% perl Makefile.PL `cat ~/.mod_perl_build_options`
% make && make test && make install
```

But mod_perl has a standard method to perform this trick. If a file named *makepl_args.mod_perl* is found in the same directory as the mod_perl build location with any of these options, it will be read in by *Makefile.PL*. Parameters supplied at the command line will override the parameters given in this file.

```
% ls -1 /usr/src
apache_1.3.xx/
makepl_args.mod_perl
mod_perl-1.xx/

% cat makepl_args.mod_perl
APACHE_SRC=../apache_1.3.xx/src DO_HTTPD=1 USE_APACI=1 \
EVERYTHING=1

% cd mod_perl-1.xx
% perl Makefile.PL
% make && make test && make install
```

Now the parameters from *makepl_args.mod_perl* file will be used, as if they were directly typed in.

Notice that this file can be located in your home directory or in the ../ directory relative to the mod_perl distribution directory. This file can also start with dot (*.makepl_args.mod_perl*) so you can keep it nicely hidden along with the rest of the dot files in your home directory.

There is a sample *makepl_args.mod_perl* in the *eg/* directory of the mod_perl distribution package, in which you might find a few options to enable experimental features to play with too!

If you are faced with a compiled Apache and no trace of the parameters used to build it, you can usually still find them if the sources were not `make clean`'d. You will find the Apache specific parameters in `apache_1.3.xx/config.status` and the mod_perl parameters in `mod_perl-1.xx/apaci/mod_perl.config`.

1.3.1.4 Discovering Whether Some Option Was Configured

mod_perl version 1.25 has introduced `Apache::Myconfig`, which provides access to the various hooks and features set when mod_perl is built. This circumvents the need to set up a live server just to find out if a certain callback hook is available.

To see whether some feature was built in or not, check the `%Apache::MyConfig::Setup` hash. For example after installing mod_perl with the following options:

```
panic% perl Makefile.PL EVERYTHING=1
```

but on the next day we don't remember what callback hooks were enabled, and we want to know whether `PERL_LOG` callback hook is enabled. One of the ways to find this out is to run the following code:

```
panic% perl -MApache::MyConfig \
-e 'print $Apache::MyConfig::Setup{PERL_LOG}'
1
```

which prints '1'--meaning that `PERL_LOG` callback hook was enabled. (That's because `EVERYTHING=1` enables them all.)

Another approach is to configure `Apache::Status` and run http://localhost/perl-status?hooks to check for enabled hooks.

You also may try to look at the symbols inside the httpd executable with help of `nm(1)` or a similar utility. For example if you want to see whether you enabled `PERL_LOG=1` while building mod_perl, we can search for a symbol with the same name but in lowercase:

```
panic% nm httpd | grep perl_log
08071724 T perl_logger
```

Indeed we can see that in our example `PERL_LOG=1` was enabled. But this will only work if you have an unstripped httpd binary. By default, `make install` strips the binary before installing it. Use the `--without-execstrip ./Configure` option to prevent stripping during *make install* phase.

Yet another approach that will work in most of the cases is to try to use the feature in question. If it wasn't configured Apache will give an error message

1.3.1.5 Using an Alternative Configuration File

By default mod_perl provides its own copy of the *Configuration* file to Apache's `./configure` utility. If you wish to pass it your own version, do this:

```
% perl Makefile.PL CONFIG=Configuration.custom
```

Where *Configuration.custom* is the pathname of the file *relative to the Apache source tree you build against*.

1.3.1.6 perl Makefile.PL Troubleshooting

1.3.1.6.1 "A test compilation with your Makefile configuration failed..."

When you see this during the `perl Makefile.PL` stage:

```
** A test compilation with your Makefile configuration
** failed. This is most likely because your C compiler
** is not ANSI. Apache requires an ANSI C Compiler, such
** as gcc. The above error message from your compiler
** will also provide a clue.
 Aborting!
```

you've got a problem with your compiler. It is possible that it's improperly installed or not installed at all. Sometimes the reason is that your Perl executable was built on a different machine, and the software installed on your machine is not the same. Generally this happens when you install the prebuilt packages, like RPM or deb. The dependencies weren't properly defined in the Perl binary package and you were allowed to install it, although some essential package is not installed.

The most frequent pitfall is a missing gdbm library. See Missing or Misconfigured libgdbm.so for more info.

But why guess, when we can actually see the real error message and understand what the real problem is. To get a real error message, edit the Apache *src/Configure* script. Down around line 2140 you will see a line like this:

```
if ./helpers/TestCompile sanity; then
```

change it to:

```
if ./helpers/TestCompile -v sanity; then
```

and try again. Now you should get a useful error message.

1.3.1.6.2 *Missing or Misconfigured libgdbm.so*

On some Linux RedHat systems you might encounter a problem during the `perl Makefile.PL` stage, when the installed from the rpm package Perl was built with the `gdbm` library, but the library isn't actually installed. If this is your situation make sure you install it before proceeding with the build process.

You can check how Perl was built by running the `perl -V` command:

```
% perl -V | grep libs
```

On my machine I get:

```
libs=-lnsl -lndbm -lgdbm -ldb -ldl -lm -lc -lposix -lcrypt
```

Sometimes the problem is even more obscure: you do have `libgdbm` installed but it's not properly installed. Do this:

```
% ls /usr/lib/libgdbm.so*
```

If you get at least three lines like I do:

```
lrwxrwxrwx   /usr/lib/libgdbm.so -> libgdbm.so.2.0.0
lrwxrwxrwx   /usr/lib/libgdbm.so.2 -> libgdbm.so.2.0.0
-rw-r--r--   /usr/lib/libgdbm.so.2.0.0
```

you are all set. On some installations the *libgdbm.so* symbolic link is missing, so you get only:

```
lrwxrwxrwx   /usr/lib/libgdbm.so.2 -> libgdbm.so.2.0.0
-rw-r--r--   /usr/lib/libgdbm.so.2.0.0
```

To fix this problem add the missing symbolic link:

```
% cd /usr/lib
% ln -s libgdbm.so.2.0.0 libgdbm.so
```

Now you should be able to build mod_perl without any problems.

Note that you might need to prepare this symbolic link as well:

```
lrwxrwxrwx   /usr/lib/libgdbm.so.2 -> libgdbm.so.2.0.0
```

with:

```
% ln -s libgdbm.so.2.0.0 libgdbm.so.2
```

Of course if when you read this a new version of the `libgdbm` library will be released, you will have to adjust the version numbers. We didn't use the usual *xx* version replacement here, to make it easier to understand how the symbolic links should be set.

1.3.1.6.3 About gdbm, db and ndbm libraries

Both the gdbm and db libraries offer ndbm emulation, which is the interface that Apache actually uses, so when you build mod_perl you end up with whichever library was linked first by the perl compile. If you build apache without mod_perl you end up with whatever appears to be your ndbm library which will vary between systems, and especially Linux distributions. You may have to work a bit to get both Apache and Perl to use the same library and you are likely to have trouble copying the dbm file from one system to another or even using it after an upgrade.

1.3.1.6.4 Undefined reference to 'PL_perl_destruct_level'

When manually building mod_perl using the shared library:

```
cd mod_perl-1.xx
perl Makefile.PL PREP_HTTPD=1
make
make test
make install

cd ../apache_1.3.xx
./configure --with-layout=RedHat --target=perlhttpd
--activate-module=src/modules/perl/libperl.a
```

you might get:

```
gcc -c  -I./os/unix -I./include   -DLINUX=2 -DTARGET=\"perlhttpd\" -DUSE_HSREGEX
-DUSE_EXPAT -I./lib/expat-lite './apaci' buildmark.c
gcc  -DLINUX=2 -DTARGET=\"perlhttpd\" -DUSE_HSREGEX -DUSE_EXPAT
-I./lib/expat-lite './apaci'     \
      -o perlhttpd buildmark.o modules.o modules/perl/libperl.a
modules/standard/libstandard.a main/libmain.a ./os/unix/libos.a ap/libap.a
regex/libregex.a lib/expat-lite/libexpat.a  -lm -lcrypt
modules/perl/libperl.a(mod_perl.o): In function 'perl_shutdown':
mod_perl.o(.text+0xf8): undefined reference to 'PL_perl_destruct_level'
mod_perl.o(.text+0x102): undefined reference to 'PL_perl_destruct_level'
mod_perl.o(.text+0x10c): undefined reference to 'PL_perl_destruct_level'
mod_perl.o(.text+0x13b): undefined reference to 'Perl_av_undef'
[more errors snipped]
```

This happens when you have a statically linked perl build (i.e. without a shared *libperl.so* library). Build a dynamically linked perl (with *libperl.so*) and the problem will disappear. See the "Building a shared Perl library" section from the *INSTALL* file that comes with Perl source.

Also see "Chapter 15.4 - Perl Build Options" from Practical mod_perl.

1.3.1.6.5 Further notes on libperl.(a/so)

Library files such as *libfoo.a* are archives that are used at linking time - these files are completely included in the final application that linked it.

Whereas *libfoo.so* indicates that it's a shared library. At the linking time the application only knows which library it wants. Only at the loading time (runtime) that shared library will be loaded.

One of the benefits of using a shared library, is that it's loaded only once. If there are two application linking to *libperl.so* that run at the same time, only the first application will need to load it. The second application will share that loaded library (that service is provided by the OS kernel). In the case of static *libfoo.a*, it'll be loaded as many times as there are applications that included it, thus consuming more memory. Of course this is not the only benefit of using shared libs.

In mod_perl 1.0, the library file is unfortunately named *libperl.(so/a)*. So you have *libperl.(so/a)* which is perl, and you have *libperl.(so/a)* which is modperl. You are certainly looking at the modperl version of *libperl.a* if you find it in the apache directory. perl's *libperl.(so/a)* lives under the perl tree (e.g. in *5.8.6/i686-linux/CORE/libperl.so*).

Some distributions (notably Debian) have chosen to put *libperl.so* and *libperl.a* into the global library loader path (e.g., */usr/lib*) which will cause linking problems when compiling mod_perl (if compiling against static perl), in which case you should move aside the *libperl.a* while building mod_perl or else will likely encounter further errors. If building against the dynamic perl's *libperl.so*, you may have similar problems but at startup time. It's the best to get rid of perl that installs its libs into */usr/lib* (or similar) and reinstall a new perl, which puts its library under the perl tree. Also see libperl.so and libperl.a.

1.3.2 mod_perl Building (make)

After completing the configuration you build the server, by calling:

```
% make
```

which compiles the source files and creates an httpd binary and/or a separate library for each module, which can either be inserted into the httpd binary when make is called from the Apache source directory or loaded later, at run time.

Note: don't put the mod_perl directory inside the Apache directory. This confuses the build process.

1.3.2.1 make Troubleshooting

1.3.2.1.1 Undefined reference to 'Perl_newAV'

This and similar error messages may show up during the make process. Generally it happens when you have a broken Perl installation. Make sure it's not installed from a broken RPM or another binary package. If it is, build Perl from source or use another properly built binary package. Run perl -V to learn what version of Perl you are using and other important details.

1.3.2.1.2 Unrecognized format specifier for...

This error usually reported due to the problems with some versions of SFIO library. Try to use the latest version to get around this problem. Or if you don't really need SFIO, rebuild Perl without this library.

1.3.3 Built Server Testing (make test)

After building the server, it's a good idea to test it thoroughly, by calling:

```
% make test
```

Fortunately mod_perl comes with a bunch of tests, which attempt to use all the features you asked for at the configuration stage. If any of the tests fails, the `make test` stage will fail.

Running `make test` will start a freshly built httpd on port 8529 running under the uid and gid of the `perl Makefile.PL` process. The httpd will be terminated when the tests are finished.

Each file in the testing suite generally includes more than one test, but when you do the testing, the program will only report how many tests were passed and the total number of tests defined in the test file. However if only some of the tests in the file fail you want to know which ones failed. To gain this information you should run the tests in verbose mode. You can enable this mode by using the `TEST_VERBOSE` parameter:

```
% make test TEST_VERBOSE=1
```

To change the default port (8529) used for the test do this:

```
% perl Makefile.PL PORT=xxxx
```

To start the newly built Apache:

```
% make start_httpd
```

To shutdown Apache:

```
% make kill_httpd
```

NOTE to Ben-SSL users: httpsd does not seem to handle */dev/null* as the location of certain files (for example some of the configuration files mentioned in *httpd.conf* can be ignored by reading them from */dev/null*) so you'll have to change these by hand. The tests are run with the `SSLDisable` directive.

1.3.3.1 Manual Testing

Tests are invoked by running the `./TEST` script located in the *./t* directory. Use the *-v* option for verbose tests. You might run an individual test like this:

```
% t/TEST -v modules/file.t
```

or all tests in a test sub-directory:

```
% t/TEST modules
```

The `TEST` script starts the server before the test is executed. If for some reason it fails to start, use `make start_httpd` to start it manually.

1.3.3.2 make test Troubleshooting

1.3.3.2.1 make test fails

You cannot run `make test` before you build Apache, so if you told `perl Makefile.PL` not to build the httpd executable, there is no httpd to run the test against. Go to the Apache source tree and run `make`, then return to the mod_perl source tree and continue with the server testing.

1.3.3.2.2 mod_perl.c is incompatible with this version of Apache

If you had a stale old Apache header layout in one of the *include* paths during the build process you will see this message when you try to execute httpd. Run the `find` (or `locate`) utility in order to locate the file *ap_mmn.h*. Delete it and rebuild Apache. RedHat installed a copy of */usr/local/include/ap_mmn.h* on my system.

For all RedHat fans, before you build Apache yourself, do:

```
% rpm -e apache
```

to remove the pre-installed RPM package first!

Users with apt systems would do:

```
% apt-get remove apache
```

instead.

1.3.3.2.3 make test......skipping test on this platform

While doing `make test` you will notice that some of the tests are reported as *skipped*. The reason is that you are missing some optional modules for these test to be passed. For a hint you might want to peek at the content of each test (you will find them all in the `./t` directory (mnemonic - t, tests). I'll list a few examples, but of course things may change in the future.

```
modules/cookie......skipping test on this platform
modules/request.....skipping test on this platform
```

Install libapreq package which includes among others the `Apache::Request` and `Apache::Cookie` modules.

```
modules/psections...skipping test on this platform
```

Install `Devel::Symdump` and `Data::Dumper`

```
modules/sandwich....skipping test on this platform
```

Install Apache::Sandwich

```
modules/stage.......skipping test on this platform
```

Install Apache::Stage

```
modules/symbol......skipping test on this platform
```

Install Devel::Symdump

Chances are that all of these are installed if you use `CPAN.pm` to `install Bundle::Apache`.

1.3.3.2.4 *make test Fails Due to Misconfigured localhost Entry*

The `make test` suite uses *localhost* to run the tests that require a network. Make sure you have this entry in */etc/hosts*:

```
127.0.0.1       localhost.localdomain   localhost
```

Also make sure that you have the loopback device [lo] configured. [Hint: try 'ifconfig lo' to test for its existence.]

1.3.4 Installation (make install)

After testing the server, the last step left is to install it. First install all the Perl side files:

```
% make install
```

Then go to the Apache source tree and complete the Apache installation (installing the configuration files, httpd and utilities):

```
% cd ../apache_1.3.xx
% make install
```

Now the installation should be considered complete. You may now configure your server and start using it.

1.3.5 Building Apache and mod_perl by Hand

If you wish to build httpd separately from mod_perl, you should use the `NO_HTTPD=1` option during the `perl Makefile.PL` (mod_perl build) stage. Then you will need to configure various things by hand and proceed to build Apache. You shouldn't run `perl Makefile.PL` before following the steps described in this section.

If you choose to manually build mod_perl, there are three things you may need to set up before the build stage:

* **mod_perl's Makefile**

  When `perl Makefile.PL` is executed, *$APACHE_SRC/modules/perl/Makefile* may need to be modified to enable various options (e.g. `ALL_HOOKS=1`).

Optionally, instead of tweaking the options during `perl Makefile.PL` you may edit *mod_perl-1.xx/src/modules/perl/Makefile* before running `perl Makefile.PL`.

- **Configuration**

  Add to *apache_1.3.xx/src/Configuration* :

  ```
  AddModule modules/perl/libperl.a
  ```

  We suggest you add this entry at the end of the *Configuration* file if you want your callback hooks to have precedence over core handlers.

  Add the following to `EXTRA_LIBS`:

  ```
  EXTRA_LIBS=`perl -MExtUtils::Embed -e ldopts`
  ```

  Add the following to `EXTRA_CFLAGS`:

  ```
  EXTRA_CFLAGS=`perl -MExtUtils::Embed -e ccopts`
  ```

- **mod_perl Source Files**

  Return to the mod_perl directory and copy the mod_perl source files into the apache build directory:

  ```
  % cp -r src/modules/perl apache_1.3.xx/src/modules/
  ```

When you have done with the configuration parts, run:

```
% perl Makefile.PL NO_HTTPD=1 DYNAMIC=1  EVERYTHING=1\
 APACHE_SRC=../apache_1.3.xx/src
```

`DYNAMIC=1` enables a build of the shared mod_perl library. Add other options if required.

```
% make install
```

Now you may proceed with the plain Apache build process. Note that in order for your changes to the *apache_1.3.xx/src/Configuration* file to take effect, you must run `apache_1.3.xx/src/Configure` instead of the default *apache_1.3.xx/configure* script:

```
% cd ../apache_1.3.xx/src
% ./Configure
% make
% make install
```

1.4 Installation Scenarios for Standalone mod_perl

There are various ways available to build Apache with the new hybrid build environment (using `USE_APACI=1`):

1.4.1 The All-In-One Way

If your goal is just to build and install Apache with mod_perl out of their source trees and have no special interest in further adjusting or enhancing Apache proceed as before:

```
% tar xzvf apache_1.3.xx.tar.gz
% tar xzvf mod_perl-1.xx.tar.gz
% cd mod_perl-1.xx
% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
% make && make test && make install
% cd ../apache_1.3.xx
% make install
```

This builds Apache statically with mod_perl, installs Apache under the default `/usr/local/apache` tree and mod_perl into the `site_perl` hierarchy of your existing Perl installation. All in one step.

1.4.2 The Flexible Way

This is the normal situation where you want to be flexible while building. Statically building mod_perl into the Apache binary (`httpd`) but via different steps, so you have a chance to include other third-party Apache modules, etc.

1. **Prepare the Apache source tree**

   The first step is as before, extract the distributions:

   ```
   % tar xvzf apache_1.3.xx.tar.gz
   % tar xzvf mod_perl-1.xx.tar.gz
   ```

2. **Install mod_perl's Perl-side and prepare the Apache-side**

   The second step is to install the Perl-side of mod_perl into the Perl hierarchy and prepare the `src/modules/perl/` subdirectory inside the Apache source tree:

   ```
   $ cd mod_perl-1.xx
   $ perl Makefile.PL \
       APACHE_SRC=../apache_1.3.xx/src \
       NO_HTTPD=1 \
       USE_APACI=1 \
       PREP_HTTPD=1 \
       EVERYTHING=1 \
       [...]
   $ make
   $ make install
   $ cd ..
   ```

   The `APACHE_SRC` option sets the path to your Apache source tree, the `NO_HTTPD` option forces this path and only this path to be used, the `USE_APACI` option triggers the new hybrid build environment and the `PREP_HTTPD` option forces preparation of the `APACHE_SRC/modules/perl/` tree but no automatic build.

Then the configuration process prepares the Apache-side of mod_perl in the Apache source tree but doesn't touch anything else in it. It then just builds the Perl-side of mod_perl and installs it into the Perl installation hierarchy.

Important: If you use `PREP_HTTPD` as described above, to complete the build you must go into the Apache source directory and run `make` and `make install`.

3. **Additionally prepare other third-party modules**

Now you have a chance to prepare third-party modules. For instance the PHP language can be added in a manner similar to the mod_perl procedure.

4. **Build the Apache Package**

Finally it's time to build the Apache package and thus also the Apache-side of mod_perl and any other third-party modules you've prepared:

```
$ cd apache_1.3.xx
$ ./configure \
    --prefix=/path/to/install/of/apache \
    --activate-module=src/modules/perl/libperl.a \
    [...]
$ make
$ make install
```

The `--prefix` option is needed if you want to change the default target directory of the Apache installation and the `--activate-module` option activates mod_perl for the configuration process and thus also for the build process. If you choose `--prefix=/usr/share/apache` the Apache directory tree will be installed in */usr/share/apache*.

Note that the files activated by `--activate-module` do not exist at this time. They will be generated during compilation.

The last three steps build, test and install the Apache-side of the mod_perl enabled server. Presumably your new server includes third-party components, otherwise you probably won't choose this method of building.

The method described above enables you to insert mod_perl into Apache without having to mangle the Apache source tree for mod_perl. It also gives you the freedom to add third-party modules.

1.4.3 When DSO can be Used

Perl versions prior to 5.6.0, built with `-Dusemymalloc`, and versions 5.6.0 and newer, built with `-Dusemymalloc` and `-Dbincompat5005`, pollute the main `httpd` program with *free* and *malloc* symbols. When `httpd` restarts (happens at startup too), any references in the main program to *free* and *malloc* become invalid, and this causes memory leaks and segfaults.

To determine if you can use a DSO mod_perl with your version of Perl, first find out which malloc your Perl was built with by running:

```
% perl -V:usemymalloc
```

If you get:

```
usemymalloc='n';
```

then it means that Perl is using the system malloc, so mod_perl will work fine as DSO.

If you get:

```
usemymalloc='y';
```

it means that Perl is using its own malloc. If you are running Perl older than 5.6.0, you *must* rebuild Perl with -Uusemymalloc in order to use it with a DSO mod_perl. If you are running Perl 5.6.0 or newer, you must either rebuild Perl with -Uusemymalloc, or make sure that binary compatibility with Perl 5.005 turned off. To find out, run:

```
% perl -V:bincompat5005
```

If you get:

```
bincompat5005='define';
```

then you *must* either rebuild Perl with -Ubincompat5005 or with -Uusemymalloc to use it with a DSO mod_perl. We recommend that you rebuild Perl with -Ubincompat5005 if Perl's malloc is a better choice for your OS.

Note that mod_perl's build system issues a warning about this problem.

If you needed to rebuild Perl don't forget to rebuild mod_perl too.

1.4.4 Build mod_perl as a DSO inside the Apache Source Tree via APACI

We have already said that the new mod_perl build environment (USE_APACI) is a hybrid. What does it mean? It means for instance that the same src/modules/perl/ stuff can be used to build mod_perl as a DSO or not, without having to edit anything especially for this. When you want to build libperl.so all you have to do is to add one single option to the above steps.

1.4.4.1 libperl.so and libperl.a

libmodperl.so would be more correct for the mod_perl file, but the name has to be libperl.so because of prehistoric Apache issues. Don't confuse the libperl.so for mod_perl with the file of the same name which comes with Perl itself. They are two different things. It is unfortunate that they happen to have the same name.

There is also a `libperl.a` which comes with the Perl installation. That's different too.

You have two options here, depending on which way you have chosen above: If you choose the All-In-One way from above then add

```
USE_DSO=1
```

to the `perl Makefile.PL` options. If you choose the Flexible way then add:

```
--enable-shared=perl
```

to Apache's `./configure` options.

As you can see only an additional `USE_DSO=1` or `--enable-shared=perl` option is needed. Everything else is done automatically: `mod_so` is automatically enabled, the Makefiles are adjusted automatically and even the `install` target from APACI now additionally installs `libperl.so` into the Apache installation tree. And even more: the `LoadModule` and `AddModule` directives (which dynamically load and insert mod_perl into httpd) are automatically added to *httpd.conf*.

1.4.5 Build mod_perl as a DSO outside the Apache Source Tree via APXS

Above we've seen how to build mod_perl as a DSO *inside* the Apache source tree. But there is a nifty alternative: building mod_perl as a DSO *outside* the Apache source tree via the new Apache 1.3 support tool `apxs` (APache eXtension). The advantage is obvious: you can extend an already installed Apache with mod_perl even if you don't have the sources (for instance, you may have installed an Apache binary package from your vendor).

Here are the build steps:

```
% tar xzvf mod_perl-1.xx.tar.gz
% cd mod_perl-1.xx
% perl Makefile.PL \
  USE_APXS=1 \
  WITH_APXS=/path/to/bin/apxs \
  EVERYTHING=1 \
   [...]
% make && make test && make install
```

This will build the DSO `libperl.so` *outside* the Apache source tree with the new Apache 1.3 support tool `apxs` and install it into the existing Apache hierarchy.

1.5 Installation Scenarios for mod_perl and Other Components

([ReaderMETA]: Please send more scenarios of mod_perl + other components installation guidelines. Thanks!)

You have now seen very detailed installation instructions for specific cases, but since mod_perl is used with many other components that plug into Apache, you will definitely want to know how to build them together with mod_perl.

Since all the steps are simple, and assuming that you now understand how the build process works, I'll show only the commands to be executed with no comments unless there is something we haven't discussed before.

Generally every example that I'm going to show consist of:

1. downloading the source distributions of the components to be used

2. un-packing them

3. configuring them

4. building Apache using the parameters appropriate to each component

5. `make test` and `make install`.

All these scenarios were tested on a Linux platform, you might need to refer to the specific component's documentation if something doesn't work for you as described below. The intention of this section is not to show you how to install other non-mod_perl components alone, but how to do this in a bundle with mod_perl.

Also, notice that the links I've used below are very likely to have changed by the time you read this document. That's why I have used the *x.x.x* convention, instead of using hardcoded version numbers. Remember to replace the *xx* place-holders with the version numbers of the distributions you are about to use. To find out the latest stable version number, visit the components' sites. So if the instructions say:

```
http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
```

go to http://perl.apache.org/download/ in order to learn the version number of the latest stable release and download the appropriate file.

Unless otherwise noted, all the components install themselves into a default location. When you run `make install` the installation program tells you where it's going to install the files.

1.5.1 *mod_perl and mod_ssl (+openssl)*

mod_ssl provides strong cryptography for the Apache 1.3 webserver via the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols by the help of the Open Source SSL/TLS toolkit OpenSSL, which is based on SSLeay from Eric A. Young and Tim J. Hudson.

Download the sources:

```
% lwp-download http://www.apache.org/dist/apache_1.3.xx.tar.gz
% lwp-download http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
% lwp-download http://www.modssl.org/source/mod_ssl-x.x.x-x.x.x.tar.gz
% lwp-download http://www.openssl.org/source/openssl-x.x.x.tar.gz
```

Un-pack:

```
% tar xvzf mod_perl-1.xx
% tar xvzf apache_1.3.xx.tar.gz
% tar xvzf mod_ssl-x.x.x-x.x.x.tar.gz
% tar xvzf openssl-x.x.x.tar.gz
```

Configure, build and install openssl:

```
% cd openssl-x.x.x
% ./config
% make && make test && make install
```

Configure mod_ssl:

```
% cd mod_ssl-x.x.x-x.x.x
% ./configure --with-apache=../apache_1.3.xx
```

Configure mod_perl:

```
% cd ../mod_perl-1.xx
% perl Makefile.PL USE_APACI=1 EVERYTHING=1 \
      DO_HTTPD=1 SSL_BASE=/usr/local/ssl \
      APACHE_PREFIX=/usr/local/apachessl \
      APACHE_SRC=../apache_1.3.xx/src \
      APACI_ARGS='--enable-module=ssl,--enable-module=rewrite'
```

Note: Do not forget that if you use `csh` or `tcsh` you may need to put all the arguments to 'perl Make-file.PL' on a single command line.

Note: If when specifying `SSL_BASE=/usr/local/ssl` Apache's configure doesn't find the ssl libraries, please refer to the mod_ssl documentation to figure out what `SSL_BASE` argument it expects (usually needed when ssl is not installed in the standard places). This topic is out of scope of this document. For some setups using `SSL_BASE=/usr/local` does the trick.

Build, test and install:

```
% make && make test && make install
% cd ../apache_1.3.xx
% make certificate
% make install
```

Now proceed with the mod_ssl and mod_perl parts of the server configuration before starting the server.

When the server starts you should see the following or similar in the *error_log* file:

```
[Fri Nov 12 16:14:11 1999] [notice] Apache/1.3.9 (Unix)
mod_perl/1.21_01-dev mod_ssl/2.4.8 OpenSSL/0.9.4 configured
-- resuming normal operations
```

1.5.2 mod_perl and mod_ssl Rolled from RPMs

As in the previous section this shows an installation of mod_perl and mod_ssl, but this time using sources/binaries prepackaged in RPMs.

As always, replace *xx* with the proper version numbers. And replace i386 with the identifier for your platform if it is different.

1.
    ```
    % get apache-mod_ssl-x.x.x.x-x.x.x.src.rpm
    ```

    Source: http://www.modssl.org

2.
    ```
    % get openssl-x.x.x.i386.rpm
    ```

    Source: http://www.openssl.org/

3.
    ```
    % lwp-download http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
    ```

    Source: http://apache.org/dist/perl

4.
    ```
    % lwp-download http://www.engelschall.com/sw/mm/mm-x.x.xx.tar.gz
    ```

    Source: http://www.engelschall.com/sw/mm/

5.
    ```
    % rpm -ivh openssl-x.x.x.i386.rpm
    ```

6.
    ```
    % rpm -ivh apache-mod_ssl-x.x.x.x-x.x.x.src.rpm
    ```

7.
    ```
    % cd /usr/src/redhat/SPECS
    ```

8.
    ```
    % rpm -bp apache-mod_ssl.spec
    ```

9.

```
% cd /usr/src/redhat/BUILD/apache-mod_ssl-x.x.x.x-x.x.x
```

10.

```
% tar xvzf mod_perl-1.xx.tar.gz
```

11.

```
% cd mod_perl-1.xx
```

12.

```
% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
   DO_HTTPD=1 \
   USE_APACI=1 \
   PREP_HTTPD=1 \
   EVERYTHING=1
```

Add or remove parameters if appropriate.

13.

```
% make
```

14.

```
% make install
```

15.

```
% cd ../mm-x.x.xx/
```

16.

```
% ./configure --disable-shared
```

17.

```
% make
```

18.

```
% cd ../mod_ssl-x.x.x-x.x.x
```

19.

```
% ./configure \
     --with-perl=/usr/bin/perl \
     --with-apache=../apache_1.3.xx\
     --with-ssl=SYSTEM \
     --with-mm=../mm-x.x.x \
     --with-layout=RedHat \
     --disable-rule=WANTHSREGEX \
     --enable-module=all \
     --enable-module=define \
     --activate-module=src/modules/perl/libperl.a \
     --enable-shared=max \
     --disable-shared=perl
```

20.

```
% make
```

21.

```
% make certificate
```

with whatever option is suitable to your configuration.

22.

```
% make install
```

You should be all set.

Note: If you use the standard config for mod_ssl don't forget to run Apache like this:

```
% httpd -DSSL
```

1.5.3 mod_perl and apache-ssl (+openssl)

Apache-SSL is a secure Webserver, based on Apache and SSLeay/OpenSSL. It is licensed under a BSD-style license which means, in short, that you are free to use it for commercial or non-commercial purposes, so long as you retain the copyright notices.

Download the sources:

```
% lwp-download http://www.apache.org/dist/apache_1.3.xx.tar.gz
% lwp-download http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
% lwp-download http://www.apache-ssl.org/.../apache_1.3.xx+ssl_x.xx.tar.gz
% lwp-download http://www.openssl.org/source/openssl-x.x.x.tar.gz
```

Un-pack:

```
% tar xvzf mod_perl-1.xx
% tar xvzf apache_1.3.xx.tar.gz
% tar xvzf openssl-x.x.x.tar.gz
```

Configure and install openssl:

```
% cd openssl-x.x.x
% ./config
% make && make test && make install
```

Patch Apache with SSLeay paths

```
% cd apache_x.xx
% tar xzvf ../apache_1.3.xx+ssl_x.xx.tar.gz
% FixPatch
Do you want me to apply the fixed-up Apache-SSL patch for you? [n] y
```

Now edit the *src/Configuration* file if needed and then configure:

```
% cd ../mod_perl-1.xx
% perl Makefile.PL USE_APACI=1 EVERYTHING=1 \
      DO_HTTPD=1 SSL_BASE=/usr/local/ssl \
      APACHE_SRC=../apache_1.3.xx/src
```

Build, test and install:

```
% make && make test && make install
% cd ../apache_1.3.xx/src
% make certificate
% make install
```

Note that you might need to modify the 'make test' stage, as it takes much longer for this server to get started and `make test` waits only a few seconds for Apache to start before it times out.

Now proceed with configuration of the apache_ssl and mod_perl parts of the server configuration files, before starting the server.

1.5.4 mod_perl and Stronghold

Stronghold is a secure SSL Web server for Unix which allows you to give your web site full-strength, 128-bit encryption.

You must first build and install Stronghold without mod_perl, following Stronghold's install procedure. For more information visit http://www.c2.net/products/sh2/ .

Having done that, download the sources:

```
% lwp-download http://apache.org/dist/perl/mod_perl-1.xx.tar.gz
```

Unpack:

```
% tar xvzf mod_perl-1.xx.tar.gz
```

Configure (assuming that you have the Stronghold sources extracted at */usr/local/stronghold*):

```
% cd mod_perl-1.xx
% perl Makefile.PL APACHE_SRC=/usr/local/stronghold/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
```

Build:

```
% make
```

Before running `make test`, you must add your `StrongholdKey` to *t/conf/httpd.conf*. If you are configuring by hand, be sure to edit *src/modules/perl/Makefile* and uncomment the `#APACHE_SSL` directive.

Test and Install:

```
% make test && make install
% cd /usr/local/stronghold
% make install
```

1.5.4.1 Note For Solaris 2.5 users

There has been a report related to the REGEX library that comes with Stronghold, that after building Apache with mod_perl it would produce core dumps. To work around this problem, in *$STRONGHOLD/src/Configuration* change:

```
Rule WANTHSREGEX=default
```

to:

```
Rule WANTHSREGEX=no
```

1.5.5 mod_perl and mod_php

This is a simple installation scenario of the mod_perl and mod_php in Apache server:

1. Configure Apache.

   ```
   % cd apache_1.3.xx
   % ./configure --prefix=/usr/local/etc/httpd
   ```

   (this step might be redundant with the recent versions of mod_php, but it's harmless.)

2. Build mod_perl.

   ```
   % cd ../mod_perl-1.xx
   % perl Makefile.PL APACHE_SRC=../apache_1.3.xxx/src NO_HTTPD=1 \
     USE_APACI=1 PREP_HTTPD=1 EVERYTHING=1
   % make
   ```

3. Build mod_php.

   ```
   % cd ../php-x.x.xx
   % ./configure --with-apache=../apache_1.3.xxx \
      --with-mysql --enable-track-vars
   % make
   % make install
   ```

4. Build Apache:

   ```
   % cd ../apache_1.3.xxx
   % ./configure --prefix=/usr/local/etc/httpd \
     --activate-module=src/modules/perl/libperl.a \
     --activate-module=src/modules/php4/libphp4.a \
     --enable-module=stats \
     --enable-module=rewrite
   % make
   ```

Note: *libperl.a* and *libphp4.a* do not exist at this time. They will be generated during compilation.

5. Test and install mod_perl

```
% cd ../mod_perl-1.xx
% make test
# make install.
```

6. Complete the Apache installation.

```
% cd ../apache_1.3.xxx
# make install
```

Note: If you need to build mod_ssl as well, make sure that you add the mod_ssl first.

1.6 mod_perl Installation with the CPAN.pm Interactive Shell

Installation of mod_perl and all the required packages is much easier with help of the `CPAN.pm` module, which provides you among other features with a shell interface to the CPAN repository. CPAN is the Comprehensive Perl Archive Network, a repository of thousands of Perl modules, scripts as well as a vast amount of documentation. See http://cpan.org for more information.

The first thing first is to download the Apache source code and unpack it into a directory -- the name of which you will need very soon.

Now execute:

```
% perl -MCPAN -eshell
```

If it's the first time that you have used it, `CPAN.pm` will ask you about a dozen questions to configure the module. It's quite easy to accomplish this task, and very helpful hints come along with the questions. When you are finished you will see the `CPAN` prompt:

```
cpan>
```

It can be a good idea to install a special `CPAN` bundle of modules to make using the CPAN module easier. Installation is as simple as typing:

```
cpan> install Bundle::CPAN
```

The `CPAN` shell can download mod_perl for you, unpack it, check for prerequisites, detect any missing third party modules, and download and install them. All you need to do to install mod_perl is to type at the prompt:

```
cpan> install mod_perl
```

You will see (I'll use `x.xx` as a placeholder for real version numbers, since these change very frequently):

```
Running make for DOUGM/mod_perl-1.xx.tar.gz
Fetching with LWP:
http://www.cpan.org/authors/id/DOUGM/mod_perl-1.xx.tar.gz

CPAN.pm: Going to build DOUGM/mod_perl-1.xx.tar.gz

Enter 'q' to stop search
Please tell me where I can find your apache src
[../apache_1.3.xx/src]
```

`CPAN.pm` will search for the latest Apache sources and suggest a directory. Here, unless the CPAN shell found it and suggested the right directory, you need to type the directory into which you unpacked Apache. The next question is about the `src` directory, which resides at the root level of the unpacked Apache distribution. In most cases the CPAN shell will suggest the correct directory.

```
Please tell me where I can find your apache src
[../apache_1.3.xx/src]
```

Answer yes to all the following questions, unless you have a reason not to do that.

```
Configure mod_perl with /usr/src/apache_1.3.xx/src ? [y]
Shall I build httpd in /usr/src/apache_1.3.xx/src for you? [y]
```

Now we will build Apache with mod_perl enabled. Quit the `CPAN` shell, or use another terminal. Go to the Apache sources root directory and run:

```
% make install
```

which will complete the installation by installing Apache's headers and the binary in the appropriate directories.

The only caveat of the process I've described is that you don't have control over the configuration process. Actually, that problem is easy to solve -- you can tell `CPAN.pm` to pass whatever parameters you want to `perl Makefile.PL`. You do this with `o conf makepl_arg` command:

```
cpan> o conf makepl_arg 'DO_HTTPD=1 USE_APACI=1 EVERYTHING=1'
```

Just list all the parameters as if you were passing them to the familiar `perl Makefile.PL`. If you add the `APACHE_SRC=/usr/src/apache_1.3.xx/src` and `DO_HTTPD=1` parameters, you will not be asked a single question. Of course you must give the correct path to the Apache source distribution.

Now proceed with `install mod_perl` as before. When the installation is completed, remember to unset the `makepl_arg` variable by executing:

```
cpan> o conf makepl_arg ''
```

If you have previously set `makepl_arg` to some value, before you alter it for the mod_perl installation you will probably want to save it somewhere so that you can restore it when you have finished with the mod_perl installation. To see the original value, use:

```
cpan> o conf makepl_arg
```

You can now install all the modules you might want to use with mod_perl. You install them all by typing a singe command:

```
cpan> install Bundle::Apache
```

This will install mod_perl if isn't yet installed, and many other packages such as: `ExtUtils::Embed`, `MIME::Base64`, `URI::URL`, `Digest::MD5`, `Net::FTP`, `LWP`, `HTML::TreeBuilder`, `CGI`, `Devel::Symdump`, `Apache::DB`, `Tie::IxHash`, `Data::Dumper` etc.

A helpful hint: If you have a system with all the Perl modules you use and you want to replicate them all elsewhere, and if you cannot just copy the whole `/usr/lib/perl5` directory because of a possible binary incompatibility on the other system, making your own bundle is a handy solution. To accomplish this the command `autobundle` can be used on the CPAN shell command line. This command writes a bundle definition file for all modules that are installed for the currently running perl interpreter.

With the clever bundle file you can then simply say

```
cpan> install Bundle::my_bundle
```

and after answering a few questions, go out for a coffee.

1.7 Installing on multiple machines

You may wish to build httpd once, then copy it to other machines. The Perl side of mod_perl needs the Apache headers files to compile. To avoid dragging and build Apache on all your other machines, there are a few Makefile targets to help you out:

```
% make tar_Apache
```

This will tar all files mod_perl installs in your Perl's *site_perl* directory, into a file called *Apache.tar*. You can then unpack this under the *site_perl* directory on another machine.

```
% make offsite-tar
```

This will copy all the header files from the Apache source directory which you configured mod_perl against, then it will `make dist` which creates a mod_perl-1.xx.tar.gz, ready to unpack on another machine to compile and install the Perl side of mod_perl.

If you really want to make your life easy you should use one of the more advanced packaging systems. For example, almost all Linux OS distributions use packaging tools on top of plain tar.gz, allowing you to track prerequisites for each package, and providing easy installation, upgrade and cleanup. One of the most widely-used packagers is RPM (Red Hat Package Manager). See http://www.rpm.org for more information.

All you have to do is prepare a SRPM (source distribution package), then build a binary release. This can be installed on any number of machines in a matter of seconds.

It will even work on live machines! If you have two identical machines (identical software and hardware, although depending on your setup hardware may be less critical). Let's say that one is a live server and the other is in development. You build an RPM with a mod_perl binary distribution, install it on the development machine and satisfy yourself that it is working and stable. You can then install the RPM package on the live server without any fear. Make sure that *httpd.conf* is correct, since it generally includes parameters such as hostname which are unique to the live machine.

When you have installed the package, just restart the server. It can be a good idea to keep a package of the previous system, in case something goes wrong. You can then easily remove the installed package and put the old one back.

([ReaderMETA]: Dear reader, Can you please share a step by step scenario of preparation of SRPMs for mod_perl? Thanks!!!)

1.8 using RPM and other packages to install mod_perl

[ReaderMETA]: Currently only RPM package. Please submit info about other available packages if you use such.

1.8.1 A word on mod_perl RPM packages

The virtues of RPM packages is a subject of much debate among mod_perl users. While RPMs do take the pain away from package installation and maintenance for most applications, the nuances of mod_perl make RPMs somewhat less than ideal for those just getting started. The following help and advice is for those new to mod_perl, Apache, Linux, and RPMs. If you know what you are doing, this is probably Old Hat - contributing your past experiences is, as always, welcomed by the community.

1.8.2 Getting Started

If you are new to mod_perl and are using this Guide and the Eagle Book to help you on your way, it is probably better to grab the latest Apache and mod_perl sources and compile the sources yourself. Not only will you find that this is less daunting than you suspect, but it will probably save you a few headaches down the line for several reasons.

First, given the pace at which the open source community produces software, RPMs, especially those found on distribution CDs, are often several versions out of date. The most recent version will not only be more stable, but will likely incorporate some new functionality that you will eventually want to play with.

It is also unlikely that the file system layout of an RPM package will match what you see in the Eagle Book and this Guide. If you are new to mod_perl, Apache, or both you will probably want to get familiar with the file system layout used by the examples given here before trying something non-standard.

Finally, the RPMs found on a typical distribution's CDs use mod_perl built with Apache's Dynamic Shared Objects (DSO) support. While mod_perl can be successfully used as a DSO module, it adds a layer of complexity that you may want to live without for now.

All that being said, should you still feel that rolling your own mod_perl enabled Apache server is not likely, here are a few helpful hints...

1.8.3 Compiling RPM source files

It is possible to compile the source files provided by RPM packages, but if you are using RPMs to ease mod_perl installation, that is not the way to do it. Both Apache and mod_perl RPMs are designed to be install-and-go. If you really want to compile mod_perl to your own specific needs, your best bet is to get the most recent sources from CPAN.

1.8.4 Mix and Match RPM and source

It is probably not the best idea to use a self-compiled Apache with a mod_perl RPM (or vice versa). Sticking with one format or the other at first will result in fewer headaches and more hair.

1.8.5 Installing a single apache+mod_perl RPM

If you use an Apache+mod_perl RPM, chances are `rpm -i` or `glint` (GUI for RPM) will have you up and running immediately, no compilation necessary. If you encounter problems, try downloading from another mirror site or searching http://rpmfind.net/ for a different package - there are plenty out there to choose from.

David Harris has started an effort to build better RPM/SRPM mod_perl packages. You will find the link to David's site from Binary distributions.

Features of this RPM:

- Installs mod_perl as an "add in" to the RedHat Apache package, but does not install mod_perl as a DSO.

- Includes the four header files required for building `libapreq` (`Apache::Request`)

- Distributes plain text forms of the pod documentation files that come with mod_perl.

- Checks the module magic number on the existing Apache package to see if things are compatible

Notes on this un-conventional RPM packaging of mod_perl

by David Harris <dharris (at) drh.net> on Oct 13, 1999

This package will install the mod_perl library files on your machine along with the following two Apache files:

```
/usr/lib/apache/mod_include_modperl.so
/usr/sbin/httpd_modperl
```

This package does not install a complete Apache subtree built with mod_perl, but rather just the two above files that are different for mod_perl. This conceptually thinks of mod_perl as a kind of an "add on" that we would like to add to the regular Apache tree. However, we are prevented from distributing mod_perl as an actual DSO, because it is not recommended by the mod_perl developers and various features must be turned off. So, instead, we distribute an httpd binary with mod_perl statically linked (httpd_modperl) and the special modified mod_include.so required for this binary (mod_include_modperl.so). You can use the exact same configuration files and other DSO modules, but you just "enable" the mod_perl "add on" by following the directions below.

To enable mod_perl, do the following:

```
(1) Configure /etc/rc.d/init.d/httpd to run httpd_modperl instead of
    httpd by changing the "daemon" command line.
(2) Replace mod_include.so with mod_include_modperl.so in the
    module loading section of /etc/httpd/conf/httpd.conf
(3) Uncomment the "AddModule mod_perl.c" line in /etc/httpd/conf/httpd.conf
```

Or run the following command:

```
/usr/sbin/modperl-enable on
```

and to disable mod_perl:

```
/usr/sbin/modperl-enable off
```

1.8.6 Compiling libapreq (Apache::Request) with the RH 6.0 mod_perl RPM

Libapreq provides the Apache::Request module.

Despite many reports of libapreq not working properly with various RPM packages, it is possible to integrate libapreq with mod_perl RPMs. It just requires a few additional steps.

1. Make certain you have the `apache-devel-x.x.x-x.i386.rpm` package installed. Also, download the latest version of libapreq from CPAN.

2. Install the source RPM for your mod_perl RPM and then do a build prep, (with `rpm -bp apache-devel-x.x.x-x.src.rpm`) which unpacks the sources. From there, copy the four header files (*mod_perl.h*, *mod_perl_version.h*, *mod_perl_xs.h*, and *mod_PL.h*) to `/usr/include/apache`.

   - 2.1 Get the SRPM from `somemir-ror.../redhat-x.x/SRPMS/mod_perl-1.xx-x.src.rpm`.

   - 2.2 Install the SRPM. This creates files in `/usr/src/redhat/SPECS` and `/usr/src/redhat/SOURCES`. Run:

```
% rpm -ih mod_perl-1.xx-x.src.rpm
```

- 2.3 Do a "prep" build of the package, which just unpackages the sources and applies any patches.

```
% rpm -bp /usr/src/redhat/SPECS/mod_perl.spec
Executing: %prep
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd /usr/src/redhat/BUILD
+ rm -rf mod_perl-1.19
+ /bin/gzip -dc /usr/src/redhat/SOURCES/mod_perl-1.19.tar.gz
+ tar -xf -
+ STATUS=0
+ [ 0 -ne 0 ]
+ cd mod_perl-1.19
++ /usr/bin/id -u
+ [ 0 = 0 ]
+ /bin/chown -Rf root .
++ /usr/bin/id -u
+ [ 0 = 0 ]
+ /bin/chgrp -Rf root .
+ /bin/chmod -Rf a+rX,g-w,o-w .
+ echo Patch #0:
Patch #0:
+ patch -p1 -b --suffix .rh -s
+ exit 0
```

NOTE: Steps 2.1 through 2.3 are just a fancy un-packing of the source tree that builds the RPM into /usr/src/redhat/BUILD/mod_perl-1.xx. You could unpack the mod_perl-1.xx.tar.gz file somewhere and then do the following steps on that source tree. The method shown above is more "pure" because you're grabbing the header files from the same tree that built the RPM. But this does not matter because RedHat is not patching that file. So, it might be better if you just grab the mod_perl source and unpack it to get these files. Less fuss and mess.

- 2.4 Look at the files you will copy: (this is not really a step, but useful to show)

```
% find /usr/src/redhat/BUILD/mod_perl-1.19 -name '*.h'
/usr/src/redhat/BUILD/mod_perl-1.19/src/modules/perl/mod_perl.h
/usr/src/redhat/BUILD/mod_perl-1.19/src/modules/perl/mod_perl_xs.h
/usr/src/redhat/BUILD/mod_perl-1.19/src/modules/perl/mod_perl_version.h
/usr/src/redhat/BUILD/mod_perl-1.19/src/modules/perl/perl_PL.h
```

- 2.5 Copy the files into /usr/include/apache.

```
% find /usr/src/redhat/BUILD/mod_perl-1.19 -name '*.h' \
  -exec cp {} /usr/include/apache \;
```

NOTE: You should not have to do:

```
    % mkdir /usr/include/apache
```

because that directory should be created by apache-devel.

3. Apply this patch to libapreq: http://www.davideous.com/modperl-rpm/distrib/libapreq-0.31_include.patch

4. Follow the libapreq directions as usual:

```
    % perl Makefile.PL
    % make && make test && make install
```

1.8.7 Installing separate Apache and mod_perl RPMs

If you are trying to install separate Apache and mod_perl RPMs, like those provided by the RedHat distributions, you may be in for a bit of a surprise. Installing the Apache RPM will go just fine, and http://localhost will bring up some type of web page for you. However, after installation of the mod_perl RPM, the How can I tell whether mod_perl is running test will show that Apache is not mod_perl enabled. This is because mod_perl needs to be added as a separate module using Apache's Dynamic Shared Objects.

To use mod_perl as a DSO, make the following modifications to your Apache configuration files:

```
httpd.conf:
----------
LoadModule perl_module modules/libperl.so
AddModule mod_perl.c

PerlModule Apache::Registry
Alias /perl/ /home/httpd/perl/
<Location /perl>
  SetHandler perl-script
  PerlHandler Apache::Registry
  PerlSendHeader On
  Options +ExecCGI
</Location>
```

After a complete shutdown and startup of the server, mod_perl should be up and running.

1.8.8 Testing the mod_perl API

Some people have reported that even when the server responds positively to the How can I tell whether mod_perl is running tests, the mod_perl API will not function properly. You may want to run the following script to verify the availability of the mod_perl API.

```
use strict;
my $r = shift;
$r->send_http_header('text/html');
$r->print("It worked!!!\n");
```

1.9 Installation Without Superuser Privileges

As you have already learned, mod_perl enabled Apache consists of two main components: Perl modules and Apache itself. Let's tackle the tasks one at a time.

I'll show a complete installation example using *stas* as a username, assuming that */home/stas* is the home directory of that user.

1.9.1 Installing Perl Modules into a Directory of Choice

Since without superuser permissions you aren't allowed to install modules into system directories like */usr/lib/perl5*, you need to find out how to install the modules under your home directory. It's easy.

First you have to decide where to install the modules. The simplest approach is to simulate the portion of the */* file system relevant to Perl under your home directory. Actually we need only two directories:

```
/home/stas/bin
/home/stas/lib
```

We don't have to create them, since that will be done automatically when the first module is installed. 99% of the files will go into the *lib* directory. Occasionally, when some module distribution comes with Perl scripts, these will go into the *bin* directory. This directory will be created if it doesn't exist.

Let's install the *CGI.pm* package, which includes a few other `CGI::*` modules. As usual, download the package from the CPAN repository, unpack it and `chdir` to the newly-created directory.

Now do a standard `perl Makefile.PL` to prepare a *Makefile*, but this time tell `MakeMaker` to use your Perl installation directories instead of the defaults.

```
% perl Makefile.PL PREFIX=/home/stas
```

`PREFIX=/home/stas` is the only part of the installation process which is different from usual. Note that if you don't like how `MakeMaker` chooses the rest of the directories, or if you are using an older version of it which requires an explicit declaration of all the target directories, you should do this:

```
% perl Makefile.PL PREFIX=/home/stas \
  INSTALLPRIVLIB=/home/stas/lib/perl5 \
  INSTALLSCRIPT=/home/stas/bin \
  INSTALLSITELIB=/home/stas/lib/perl5/site_perl \
  INSTALLBIN=/home/stas/bin \
  INSTALLMAN1DIR=/home/stas/lib/perl5/man  \
  INSTALLMAN3DIR=/home/stas/lib/perl5/man3
```

The rest is as usual:

```
% make
% make test
% make install
```

`make install` installs all the files in the private repository. Note that all the missing directories are created automatically, so there is no need to create them in first place. Here (slightly edited) is what it does :

```
Installing /home/stas/lib/perl5/CGI/Cookie.pm
Installing /home/stas/lib/perl5/CGI.pm
Installing /home/stas/lib/perl5/man3/CGI.3
Installing /home/stas/lib/perl5/man3/CGI::Cookie.3
Writing /home/stas/lib/perl5/auto/CGI/.packlist
Appending installation info to /home/stas/lib/perl5/perllocal.pod
```

If you have to use the explicit target parameters, instead of a single `PREFIX` parameter, you will find it useful to create a file called for example *~/.perl_dirs* (where ~ is `/home/stas` in our example) containing:

```
PREFIX=/home/stas \
INSTALLPRIVLIB=/home/stas/lib/perl5 \
INSTALLSCRIPT=/home/stas/bin \
INSTALLSITELIB=/home/stas/lib/perl5/site_perl \
INSTALLBIN=/home/stas/bin \
INSTALLMAN1DIR=/home/stas/lib/perl5/man  \
INSTALLMAN3DIR=/home/stas/lib/perl5/man3
```

From now on, any time you want to install perl modules locally you simply execute:

```
% perl Makefile.PL `cat ~/.perl_dirs`
% make
% make test
% make install
```

Using this method you can easily maintain several Perl module repositories. For example, you could have one for production Perl and another for development:

```
% perl Makefile.PL `cat ~/.perl_dirs.production`
```

or

```
% perl Makefile.PL `cat ~/.perl_dirs.develop`
```

1.9.2 Making Your Scripts Find the Locally Installed Modules

Perl modules are generally placed in four main directories. To find these directories, execute:

```
% perl -V
```

The output contains important information about your Perl installation. At the end you will see:

```
Characteristics of this binary (from libperl):
Built under linux
Compiled at Apr  6 1999 23:34:07
@INC:
  /usr/lib/perl5/5.00503/i386-linux
  /usr/lib/perl5/5.00503
  /usr/lib/perl5/site_perl/5.005/i386-linux
  /usr/lib/perl5/site_perl/5.005
  .
```

It shows us the content of the Perl special variable `@INC`, which is used by Perl to look for its modules. It is equivalent to the `PATH` environment variable in Unix shells which is used to find executable programs.

Notice that Perl looks for modules in the . directory too, which stands for the current directory. It's the last entry in the above output.

Of course this example is from version *5.00503* of Perl installed on my x86 architecture PC running Linux. That's why you see *i386-linux* and *5.00503*. If your system runs a different version of Perl, operating system, processor or chipset architecture, then some of the directories will have different names.

I also have a perl-5.6.0 installed under `/usr/local/lib/` so when I do:

```
% /usr/local/bin/perl5.6.0 -V
```

I see:

```
@INC:
  /usr/local/lib/perl5/5.6.0/i586-linux
  /usr/local/lib/perl5/5.6.0
  /usr/local/lib/site_perl/5.6.0/i586-linux
  /usr/local/lib/site_perl
```

Note that it's still *Linux*, but the newer Perl version uses the version of my Pentium processor (thus the *i586* and not *i386*). This makes use of compiler optimizations for Pentium processors when the binary Perl extensions are created.

All the platform specific files, such as compiled C files glued to Perl with `XS` or `SWIG`, are supposed to go into the `i386-linux`-like directories.

Important: As we have installed the Perl modules into non-standard directories, we have to let Perl know where to look for the four directories. There are two ways to accomplish this. You can either set the `PERL5LIB` environment variable, or you can modify the `@INC` variable in your scripts.

Assuming that we use perl-5.00503, in our example the directories are:

```
/home/stas/lib/perl5/5.00503/i386-linux
/home/stas/lib/perl5/5.00503
/home/stas/lib/perl5/site_perl/5.005/i386-linux
/home/stas/lib/perl5/site_perl/5.005
```

As mentioned before, you find the exact directories by executing `perl -V` and replacing the global Perl installation's base directory with your home directory.

Modifying `@INC` is quite easy. The best approach is to use the `lib` module (pragma), by adding the following snippet at the top of any of your scripts that require the locally installed modules.

```
use lib qw(/home/stas/lib/perl5/5.00503/
           /home/stas/lib/perl5/site_perl/5.005);
```

Another way is to write code to modify `@INC` explicitly:

```
BEGIN {
  unshift @INC,
    qw(/home/stas/lib/perl5/5.00503
       /home/stas/lib/perl5/5.00503/i386-linux
       /home/stas/lib/perl5/site_perl/5.005
       /home/stas/lib/perl5/site_perl/5.005/i386-linux);
      }
```

Note that with the `lib` module we don't have to list the corresponding architecture specific directories, since it adds them automatically if they exist (to be exact, when *$dir/$archname/auto* exists).

Also, notice that both approaches *prepend* the directories to be searched to `@INC`. This allows you to install a more recent module into your local repository and Perl will use it instead of the older one installed in the main system repository.

Both approaches modify the value of `@INC` at compilation time. The `lib` module uses the *BEGIN* block as well, but internally.

Now, let's assume the following scenario. I have installed the `LWP` package in my local repository. Now I want to install another module (e.g. mod_perl) and it has `LWP` listed in its prerequisites list. I know that I have `LWP` installed, but when I run `perl Makefile.PL` for the module I'm about to install I'm told that I don't have `LWP` installed.

There is no way for Perl to know that we have some locally installed modules. All it does is search the directories listed in `@INC`, and since the latter contains only the default four directories (plus the . directory), it cannot find the locally installed `LWP` package. We cannot solve this problem by adding code to modify `@INC`, but changing the `PERL5LIB` environment variable will do the trick. If you are using `t?csh` for interactive work, do this:

```
setenv PERL5LIB /home/stas/lib/perl5/5.00503:
/home/stas/lib/perl5/site_perl/5.005
```

It should be a single line with directories separated by colons (`:`) and no spaces. If you are a `(ba)?sh` user, do this:

```
export PERL5LIB=/home/stas/lib/perl5/5.00503:
/home/stas/lib/perl5/site_perl/5.005
```

Again make it a single line. If you use bash you can use multi-line commands by terminating split lines with a backslash (\), like this:

```
export PERL5LIB=/home/stas/lib/perl5/5.00503:\
/home/stas/lib/perl5/site_perl/5.005
```

As with `use lib`, perl automatically prepends the architecture specific directories to `@INC` if those exist.

When you have done this, verify the value of the newly configured `@INC` by executing `perl -V` as before. You should see the modified value of `@INC`:

```
% perl -V

Characteristics of this binary (from libperl):
Built under linux
Compiled at Apr  6 1999 23:34:07
%ENV:
  PERL5LIB="/home/stas/lib/perl5/5.00503:
  /home/stas/lib/perl5/site_perl/5.005"
@INC:
  /home/stas/lib/perl5/5.00503/i386-linux
  /home/stas/lib/perl5/5.00503
  /home/stas/lib/perl5/site_perl/5.005/i386-linux
  /home/stas/lib/perl5/site_perl/5.005
  /usr/lib/perl5/5.00503/i386-linux
  /usr/lib/perl5/5.00503
  /usr/lib/perl5/site_perl/5.005/i386-linux
  /usr/lib/perl5/site_perl/5.005
  .
```

When everything works as you want it to, add these commands to your *.tcshrc* or *.bashrc* file. The next time you start a shell, the environment will be ready for you to work with the new Perl.

Note that if you have a `PERL5LIB` setting, you don't need to alter the `@INC` value in your scripts. But if for example someone else (who doesn't have this setting in the shell) tries to execute your scripts, Perl will fail to find your locally installed modules. The best example is a crontab script that *might* use a different SHELL environment and therefore the `PERL5LIB` setting won't be available to it.

So the best approach is to have both the `PERL5LIB` environment variable and the explicit `@INC` extension code at the beginning of the scripts as described above.

1.9.3 The CPAN.pm Shell and Locally Installed Modules

As we saw in the section describing the usage of the `CPAN.pm` shell to install mod_perl, it saves a great deal of time. It does the job for us, even detecting the missing modules listed in prerequisites, fetching and installing them. So you might wonder whether you can use `CPAN.pm` to maintain your local repository as well.

When you start the `CPAN` interactive shell, it searches first for the user's private configuration file and then for the system wide one. When I'm logged as user `stas` the two files on my setup are:

```
/home/stas/.cpan/CPAN/MyConfig.pm
/usr/lib/perl5/5.00503/CPAN/Config.pm
```

If there is no CPAN shell configured on your system, when you start the shell for the first time it will ask you a dozen configuration questions and then create the *Config.pm* file for you.

If you've got it already system-wide configured, you should have a /usr/lib/perl5/5.00503/CPAN/Config.pm. If you have a different Perl version, alter the path to use your Perl's version number, when looking up the file. Create the directory (mkdir -p creates the whole path at once) where the local configuration file will go:

```
% mkdir -p /home/stas/.cpan/CPAN
```

Now copy the system wide configuration file to your local one.

```
% cp /usr/lib/perl5/5.00503/CPAN/Config.pm \
  /home/stas/.cpan/CPAN/MyConfig.pm
```

The only thing left is to change the base directory of *.cpan* in your local file to the one under your home directory. On my machine I replace /usr/src/.cpan (that's where my system's .cpan directory resides) with /home/stas. I use Perl of course!

```
% perl -pi -e 's|/usr/src|/home/stas|' \
/home/stas/.cpan/CPAN/MyConfig.pm
```

Now you have the local configuration file ready, you have to tell it what special parameters you need to pass when executing the perl Makefile.PL stage.

Open the file in your favorite editor and replace line:

```
'makepl_arg' => q[],
```

with:

```
'makepl_arg' => q[PREFIX=/home/stas],
```

Now you've finished the configuration. Assuming that you are logged in as the same user you have prepared the local installation for (*stas* in our example), start it like this:

```
% perl -MCPAN -e shell
```

From now on any module you try to install will be installed locally. If you need to install some system modules, just become the superuser and install them in the same way. When you are logged in as the superuser, the system-wide configuration file will be used instead of your local one.

If you have used more than just the PREFIX variable, modify *MyConfig.pm* to use them. For example if you have used these variables:

```
    perl Makefile.PL PREFIX=/home/stas \
    INSTALLPRIVLIB=/home/stas/lib/perl5 \
    INSTALLSCRIPT=/home/stas/bin \
    INSTALLSITELIB=/home/stas/lib/perl5/site_perl \
    INSTALLBIN=/home/stas/bin \
    INSTALLMAN1DIR=/home/stas/lib/perl5/man  \
    INSTALLMAN3DIR=/home/stas/lib/perl5/man3
```

then replace `PREFIX=/home/stas` in the line:

```
  'makepl_arg' => q[PREFIX=/home/stas],
```

with all the variables from above, so that the line becomes:

```
  'makepl_arg' => q[PREFIX=/home/stas \
    INSTALLPRIVLIB=/home/stas/lib/perl5 \
    INSTALLSCRIPT=/home/stas/bin \
    INSTALLSITELIB=/home/stas/lib/perl5/site_perl \
    INSTALLBIN=/home/stas/bin \
    INSTALLMAN1DIR=/home/stas/lib/perl5/man  \
    INSTALLMAN3DIR=/home/stas/lib/perl5/man3],
```

If you arrange all the above parameters in one line, you can remove the backslashes (\).

1.9.4 Making a Local Apache Installation

Just like with Perl modules, if you don't have permissions to install files into the system area you have to install them locally under your home directory. It's almost the same as a plain installation, but you have to run the server listening to a port number greater than 1024 since only root processes can listen to lower numbered ports.

Another important issue you have to resolve is how to add startup and shutdown scripts to the directories used by the rest of the system services. You will have to ask your system administrator to assist you with this issue.

To install Apache locally, all you have to do is to tell `.configure` in the Apache source directory what target directories to use. If you are following the convention that I use, which makes your home directory look like the / (base) directory, the invocation parameters would be:

```
  ./configure --prefix=/home/stas
```

Apache will use the prefix for the rest of its target directories instead of the default `/usr/local/apache`. If you want to see what they are, before you proceed add the *--show-layout* option:

```
  ./configure --prefix=/home/stas --show-layout
```

You might want to put all the Apache files under `/home/stas/apache` following Apache's convention:

```
./configure --prefix=/home/stas/apache
```

If you want to modify some or all of the names of the automatically created directories:

```
./configure --prefix=/home/stas/apache \
  --sbindir=/home/stas/apache/sbin
  --sysconfdir=/home/stas/apache/etc
  --localstatedir=/home/stas/apache/var \
  --runtimedir=/home/stas/apache/var/run \
  --logfiledir=/home/stas/apache/var/logs \
  --proxycachedir=/home/stas/apache/var/proxy
```

That's all!

Also remember that you can start the script only under a user and group you belong to. You must set the `User` and `Group` directives in *httpd.conf* to appropriate values.

1.9.5 Manual Local mod_perl Enabled Apache Installation

Now when we have learned how to install local Apache and Perl modules separately, let's see how to install mod_perl enabled Apache in our home directory. It's almost as simple as doing each one separately, but there is one wrinkle you need to know about which I'll mention at the end of this section.

Let's say you have unpacked the Apache and mod_perl sources under */home/stas/src* and they look like this:

```
% ls /home/stas/src
/home/stas/src/apache_1.3.xx
/home/stas/src/mod_perl-1.xx
```

where *xx* are the version numbers as usual. You want the Perl modules from the mod_perl package to be installed under */home/stas/lib/perl5* and the Apache files to go under */home/stas/apache*. The following commands will do that for you:

```
% perl Makefile.PL \
PREFIX=/home/stas \
APACHE_PREFIX=/home/stas/apache \
APACHE_SRC=../apache_1.3.xx/src \
DO_HTTPD=1 \
USE_APACI=1 \
EVERYTHING=1
% make && make test && make install
% cd ../apache_1.3.xx
% make install
```

If you need some parameters to be passed to the `.configure` script, as we saw in the previous section use `APACI_ARGS`. For example:

```
APACI_ARGS='--sbindir=/home/stas/apache/sbin, \
  --sysconfdir=/home/stas/apache/etc, \
  --localstatedir=/home/stas/apache/var, \
  --runtimedir=/home/stas/apache/var/run, \
  --logfiledir=/home/stas/apache/var/logs, \
  --proxycachedir=/home/stas/apache/var/proxy'
```

Note that the above multiline splitting will work only with `(ba)?sh`, `t?csh` users will have to list all the parameters on a single line.

Basically the installation is complete. The only remaining problem is the `@INC` variable. This won't be correctly set if you rely on the `PERL5LIB` environment variable unless you set it explicitly in a startup file which is `require`'d before loading any other module that resides in your local repository. A much nicer approach is to use the `lib` pragma as we saw before, but in a slightly different way--we use it in the startup file and it affects all the code that will be executed under mod_perl handlers. For example:

```
PerlRequire /home/stas/apache/perl/startup.pl
```

where `startup.pl` starts with:

```
use lib qw(/home/stas/lib/perl5/5.00503/
           /home/stas/lib/perl5/site_perl/5.005);
```

Note that you can still use the hard-coded `@INC` modifications in the scripts themselves, but be aware that scripts modify `@INC` in `BEGIN` blocks and mod_perl executes the `BEGIN` blocks only when it performs script compilation. As a result, `@INC` will be reset to its original value after the scripts are compiled and the hard-coded settings will be forgotten. See the section '@INC and mod_perl' for more information.

The only place you can alter the "original" value is during the server configuration stage either in the startup file or by putting

```
PerlSetEnv Perl5LIB \
/home/stas/lib/perl5/5.00503/:/home/stas/lib/perl5/site_perl/5.005
```

in *httpd.conf*, but the latter setting will be ignored if you use the `PerlTaintcheck` setting, and I hope you do use it.

The rest of the mod_perl configuration and use is just the same as if you were installing mod_perl as superuser.

1.9.5.1 Resource Usage

Another important thing to keep in mind is the consumption of system resources. mod_perl is memory hungry. If you run a lot of mod_perl processes on a public, multiuser machine, most likely the system administrator of this machine will ask you to use less resources and may even shut down your mod_perl server and ask you to find another home for it. You have a few options:

- Reduce resources usage (see Preventing Your Processes from Growing).

- Ask your ISP's system administrator whether they can setup a dedicated machine for you, so that you will be able to install as much memory as you need. If you get a dedicated machine the chances are that you will want to have root access, so you may be able to manage the administration yourself. Then you should consider keeping on the list of the system administrator's responsibilities the following items: a reliable electricity supply and network link. And of course making sure that the important security patches get applied and the machine is configured to be secure. Finally having the machine physically protected, so no one will turn off the power or break it.

- Look for another ISP with lots of resources or one that supports mod_perl. You can find a list of these ISPs on this site.

1.9.6 Local mod_perl Enabled Apache Installation with CPAN.pm

Again, CPAN makes installation and upgrades simpler. You have seen how to install a mod_perl enabled server using `CPAN.pm`'s interactive shell. You have seen how to install Perl modules and Apache locally. Now all you have to do is to merge these techniques into a single "local mod_perl Enabled Apache Installation with CPAN.pm" technique.

Assuming that you have configured `CPAN.pm` to install Perl modules locally, the installation is very simple. Start the `CPAN.pm` shell, set the arguments to be passed to `perl Makefile.PL` (modify the example setting to suit your needs), and tell `CPAN.pm` to do the rest for you:

```
% perl -MCPAN -eshell
cpan> o conf makepl_arg 'DO_HTTPD=1 USE_APACI=1 EVERYTHING=1 \
      PREFIX=/home/stas APACHE_PREFIX=/home/stas/apache'
cpan> install mod_perl
```

When you use `CPAN.pm` for local installations, after the mod_perl installation is complete you must make sure that the value of `makepl_arg` is restored to its original value.

The simplest way to do this is to quit the interactive shell by typing *quit* and reenter it. But if you insist here is how to make it work without quitting the shell. You really want to skip this :)

If you want to continue working with CPAN *without* quitting the shell, you must:

1. **remember the value of `makepl_arg`**
2. **change it to suit your new installation**
3. **build and install mod_perl**
4. **restore it after completing mod_perl installation**

this is quite a cumbersome task as of this writing, but I believe that `CPAN.pm` will eventually be improved to handle this more easily.

So if you are still with me, start the shell as usual:

```
% perl -MCPAN -eshell
```

First, read the value of the `makepl_arg`:

```
cpan> o conf makepl_arg

PREFIX=/home/stas
```

It will be something like `PREFIX=/home/stas` if you configured `CPAN.pm` to install modules locally. Save this value:

```
cpan> o conf makepl_arg.save PREFIX=/home/stas
```

Second, set a new value, to be used by the mod_perl installation process. (You can add parameters to this line, or remove them, according to your needs.)

```
cpan> o conf makepl_arg 'DO_HTTPD=1 USE_APACI=1 EVERYTHING=1 \
        PREFIX=/home/stas APACHE_PREFIX=/home/stas/apache'
```

Third, let `CPAN.pm` build and install mod_perl for you:

```
cpan> install mod_perl
```

Fourth, reset the original value to `makepl_arg`. We do this by printing the value of the saved variable and assigning it to `makepl_arg`.

```
cpan> o conf makepl_arg.save

PREFIX=/home/stas

cpan> o conf makepl_arg PREFIX=/home/stas
```

Not so neat, but a working solution. You could have written the value on a piece of paper instead of saving it to `makepl_arg.save`, but you are more likely to make a mistake that way.

1.10 Automating installation

- **Apache Builder**

  James G Smith wrote an Apache Builder, that can install a combination of Apache, mod_perl, and mod_ssl -- it also has limited support for including mod_php in the mix. The builder is available from James' CPAN directory: $CPAN/authors/id/J/JS/JSMITH/ in the package *build-apache-xx.tar.gz*.

- **Aphid Apache Installer**

  Aphid provides a facility for bootstrapping SSL-enabled Apache web servers (mod_ssl) with an embedded Perl interpreter (mod_perl). Source is downloaded from the Internet, compiled, and the resulting system is installed in the directory you specify.

  http://sourceforge.net/projects/aphid/

1.11 How can I tell whether mod_perl is running?

There are a few ways. In older versions of apache (< 1.3.6 ?) you could check that by running `httpd -v`, but it no longer works. Now you should use `httpd -l`. Please note that it is not enough to have it installed, you have to configure it for mod_perl and restart the server too.

1.11.1 Checking the error_log

When starting the server, just check the `error_log` file for the following message:

```
[Thu Dec  3 17:27:52 1998] [notice] Apache/1.3.1 (Unix) mod_perl/1.15 configured
                                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  -- resuming normal operations
```

1.11.2 Testing by viewing /perl-status

Assuming that you have configured the <Location /perl-status> section in the server configuration file fetch: http://www.example.com/perl-status using your favorite Mozilla browser :-)

You should see something like this:

```
Embedded Perl version 5.00503 for Apache/1.3.9 (Unix) mod_perl/1.21
process 50880, running since Mon Dec 6 14:31:45 1999
```

1.11.3 Testing via telnet

Knowing the port you have configured apache to listen on, you can use `telnet` to talk directly to it.

Assuming that your mod_perl enabled server listens to port 8080, telnet to your server at port 8080, and type `HEAD / HTTP/1.0` then press the ENTER key TWICE:

```
% telnet localhost 8080<ENTER>
HEAD / HTTP/1.0<ENTER><ENTER>
```

You should see a response like this:

```
HTTP/1.1 200 OK
Date: Mon, 06 Dec 1999 12:27:52 GMT
Server: Apache/1.3.9 (Unix) mod_perl/1.21
Connection: close
Content-Type: text/html

Connection closed.
```

The line

```
Server: Apache/1.3.9 (Unix) mod_perl/1.21
```

confirms that you have mod_perl installed and its version is `1.21`.

However, just because you have got mod_perl linked in there, that does not mean that you have configured your server to handle Perl scripts with mod_perl. You will find configuration assistance at ModPerlConfiguration

1.11.4 Testing via a CGI script

Another method is to invoke a CGI script which dumps the server's environment.

I assume that you have configured the server so that scripts running under location */perl/* are handled by the `Apache::Registry` handler and that you have the `PerlSendHeader` directive set to `On`.

Copy and paste the script below (no need for a shebang line!). Let's say you name it *test.pl*, save it at the root of the CGI scripts and CGI root is mapped directly to the */perl* location of your server.

```
print "Content-type: text/plain\r\n\r\n";
print "Server's environment\n";
foreach ( keys %ENV ) {
    print "$_\t$ENV{$_}\n";
}
```

Make it readable and executable by server (you may need to tune these permissions on a public host):

```
% chmod a+rx test.pl
```

Now fetch the URL `http://www.example.com:8080/perl/test.pl` (replace 8080 with the port your mod_perl enabled server is listening to). You should see something like this (the output has been edited):

```
SERVER_SOFTWARE Apache/1.3.10-dev (Unix) mod_perl/1.21_01-dev
GATEWAY_INTERFACE       CGI-Perl/1.1
DOCUMENT_ROOT   /home/httpd/docs
REMOTE_ADDR     127.0.0.1
[more environment variables snipped]
MOD_PERL        mod_perl/1.21_01-dev
[more environment variables snipped]
```

If you see the that the value of `GATEWAY_INTERFACE` is `CGI-Perl/1.1` everything is OK.

If there is an error you might have to add a shebang line `#!/usr/bin/perl` as a first line of the CGI script and then try it again. If you see:

```
GATEWAY_INTERFACE       CGI/1.1
```

it means that you have configured this location to run under mod_cgi and not mod_perl.

Also note that there is a `MOD_PERL` environment variable if you run under a mod_perl handler, it's set to the *mod_perl/x.xx* string, where *x.xx* is the version number of mod_perl.

Based on this difference you can write code like this:

```
BEGIN {
    # perl5.004 or better is a must under mod_perl
  require 5.004 if $ENV{MOD_PERL};
}
```

You might wonder why in the world you would need to know what handler you are running under. Well, for example you will want to use `Apache::exit()` and not `CORE::exit()` in your modules, but if you think that your script might be used in both environments (mod_cgi and mod_perl) you will have to override the `exit()` subroutine and to make decision what method to use at the runtime.

Note that if you run scripts under the `Apache::Registry` handler, it takes care of overriding the `exit()` call for you, so it's not an issue. For reasons and implementations see: Terminating requests and processes, exit() function and also Writing Mod Perl scripts and Porting plain CGIs to it.

1.11.5 Testing via lwp-request

Yet another one. Why do I show all these approaches? While here they serve a very simple purpose, they can be helpful in other situations.

Assuming you have the `libwww-perl` (LWP) package installed (you will need it installed in order to pass mod_perl's `make test` anyway):

```
% lwp-request -e -d http://www.example.com
```

Will show you all the headers. The `-d` option disables printing the response content.

```
% lwp-request -e -d http://www.example.com | egrep '^Server:'
```

To see the server version only.

Specify the port number if your server is listening to a port other than port 80. For example: `http://www.example.com:8080`.

This technique works only if `ServerTokens` directive is set to `Full` or not specified in *httpd.conf*. That's because this directive controls whether the components information is displayed or not.

1.12 General Notes

1.12.1 Is it possible to run mod_perl enabled Apache as suExec?

The answer is **No**. The reason is that you can't *"suid"* a part of a process. mod_perl lives inside the Apache process, so its UID and GID are the same as the Apache process.

You have to use mod_cgi if you need this functionality.

Another solution is to use a crontab to call some script that will check whether there is something to do and will execute it. The mod_perl script will be able to create and update this todo list.

1.12.2 Should I Rebuild mod_perl if I have Upgraded Perl?

Yes, you should. You have to rebuild the mod_perl enabled server since it has a hard-coded `@INC` variable. This points to the old Perl and it is probably linked to an old `libperl` library. If for some reason you need to keep the old Perl version around you can modify `@INC` in the startup script, but it is better to build afresh to save you getting into a mess.

1.12.3 Perl installation requirements

Make sure you have Perl installed! The latest stable version if possible. Minimum perl 5.004! If you don't have it, install it. Follow the instructions in the distribution's `INSTALL` file.

During the configuration stage (while running `./Configure`), to be able to dynamically load Perl module extensions, make sure you answer `YES` to the question:

```
  Do you wish to use dynamic loading? [y]
```

1.12.4 mod_auth_dbm nuances

If you are a `mod_auth_dbm` or `mod_auth_db` user you may need to edit Perl's `Config` module. When Perl is configured it attempts to find libraries for ndbm, gdbm, db, etc., for the DB*_File modules. By default, these libraries are linked with Perl and remembered by the `Config` module. When mod_perl is configured with apache, the `ExtUtils::Embed` module requires these libraries to be linked with httpd so Perl extensions will work under mod_perl. However, the order in which these libraries are stored in **Config.pm** may confuse `mod_auth_db*`. If `mod_auth_db*` does not work with mod_perl, take a look at the order with the following command:

```
 % perl -V:libs
```

Here's an example:

```
 libs='-lnet -lnsl_s -lgdbm -lndbm -ldb -ldld -lm -lc -lndir -lcrypt';
```

If `-lgdbm` or `-ldb` is before `-lndbm` (as it is in the example) edit *Config.pm* and move `-lgdbm` and `-ldb` to the end of the list. Here's how to find *Config.pm*:

```
 % perl -MConfig -e 'print "$Config{archlibexp}/Config.pm\n"'
```

Under Solaris, another solution for building Apache/mod_perl+mod_auth_dbm is to remove the DBM and NDBM "emulation" from *libgdbm.a*. It seems that Solaris already provides its own DBM and NDBM, and in our installation we found there's no reason to build GDBM with them.

In our Makefile for GDBM, we changed

```
  OBJS = $(DBM_OF) $(NDBM_OF) $(GDBM_OF)
```

to

```
  OBJS = $(GDBM_OF)
```

Rebuild libgdbm before Apache/mod_perl.

1.12.5 Stripping Apache to make it almost a Perl-server

Since most of the functionality that various apache mod_* modules provide is implemented in the `Apache::{*}` Perl modules, it was reported that one can build an Apache server with mod_perl only. If you can reduce the requirements to whatever mod_perl can handle, you can eliminate almost every other module. Then basically you will have a Perl-server, with C code to handle the tricky HTTP bits. The only module you will need to leave in is `mod_actions`.

1.12.6 Saving the config.status Files with mod_perl, php, ssl and Other Components

Typically, when building the bloated Apache that sits behind Squid or whatever, you need mod_perl, php, mod_ssl and the rest. As you install each they typically overwrite each other's `config.status` files. Save them after each step, so you will be able to reuse them later.

1.12.7 What Compiler Should Be Used to Build mod_perl?

All Perl modules that use C extensions must be compiled using the same compiler that your copy of Perl was built with and the same compile options.

When you run `perl Makefile.PL`, a *Makefile* is created. This *Makefile* includes the same compilation options that were used to build Perl itself. They are stored in the *Config.pm* module and can be displayed with the `Perl -V` command. All these options are re-applied when compiling Perl modules.

If you use a different compiler to build Perl extensions, chances are that the options that a different compiler uses won't be the same, or they might be interpreted in a completely different way. So the code either won't compile or it will dump core when run or maybe it will behave in most unexpected ways.

Since mod_perl uses Perl, Apache and third party modules, and they all work together, it's essential to use the same compiler while building each of the components.

You shouldn't worry about this when compiling Perl modules since Perl will choose what's right automatically. Unless you override things. If you do that, you are on your own...

Similarly, if you compile a non-Perl component separately, you should make sure to use both the same compiler and the same options used to build Perl.

1.12.8 Unescaping error_log

Starting from 1.3.30, the Apache logging API escapes everything that goes to *error_log*, therefore if you're annoyed by this feature during the development phase (as your error messages will be all messed up) you can disable the escaping during the Apache build time:

```
% CFLAGS="-DAP_UNSAFE_ERROR_LOG_UNESCAPED" ./configure ...
```

Or if you build a static perl

```
% perl Makefile.PL ... PERL_EXTRA_CFLAGS=-DAP_UNSAFE_ERROR_LOG_UNESCAPED
```

Do **not** use that CFLAGS in production unless you know what you are doing.

1.13 OS Related Notes

- Gary Shea <shea (at) xmission.com> discovered a nasty BSDI bug (seen in versions 2.1 and 3.0) related to dynamic loading and found two workarounds:

  It turns out that they use `argv[0]` to determine where to find the link tables at run-time, so if a program either changes `argv[0]`, or does a chdir() (like Apache!) it can easily confuse the dynamic loader. The short-term solutions to the problem are simple. Either of the following will work:

  1) Call httpd with a full path, e.g. /opt/www/bin/httpd

  2) Put the httpd you wish to run in a directory in your PATH *before* any other directory containing a version of httpd, then call it as 'httpd'. Don't use a relative path!

1.14 Pros and Cons of Building mod_perl as DSO

On modern Unix derivatives there is a nifty mechanism usually called dynamic linking/loading of Dynamic Shared Objects (DSO), which provides a way to build a piece of program code in a special format for loading in at run-time into the address space of an executable program.

As of Apache 1.3, the configuration system supports two optional features for taking advantage of the modular DSO approach: compilation of the Apache core program into a DSO library for shared usage and compilation of the Apache modules into DSO files for explicit loading at run-time.

Should you use this method? Read the pros and cons and decide for yourself.

Pros:

- The server package is more flexible at run-time because the actual server process can be assembled at run-time via `LoadModule` *httpd.conf* configuration commands instead of *Configuration* `AddModule` commands at build-time. For instance this way one is able to run different server instances (standard & SSL version, with and without mod_perl) with only one Apache installation.

- The server package can be easily extended with third-party modules even after installation. This is at least a great benefit for vendor package maintainers who can create an Apache core package and additional packages containing extensions like PHP4, mod_perl, mod_fastcgi, etc.

- Easier Apache module prototyping because with the DSO/apxs pair you can both work outside the Apache source tree and only need an apxs -i command followed by an apachectl restart to bring a new version of your currently developed module into the running Apache server.

Cons:

- The DSO mechanism cannot be used on every platform because not all operating systems support dynamic loading of code into the address space of a program.

- The server starts up approximately 20% slower because of the symbol resolving overhead the Unix loader now has to do.

- The server runs approximately 5% slower on some platforms because position independent code (PIC) sometimes needs complicated assembler tricks for relative addressing which are not necessarily as fast as absolute addressing.

- Because DSO modules cannot be linked against other DSO-based libraries (ld -lfoo) on all platforms (for instance a.out-based platforms usually don't provide this functionality while ELF-based platforms do) you cannot use the DSO mechanism for all types of modules. Or in other words, modules compiled as DSO files are restricted to only use symbols from the Apache core, from the C library (libc) and all other dynamic or static libraries used by the Apache core, or from static library archives (libfoo.a) containing position independent code. The only way you can use other code is to either make sure the Apache core itself already contains a reference to it, loading the code yourself via dlopen() or enabling the SHARED_CHAIN rule while building Apache when your platform supports linking DSO files against DSO libraries.

- Under some platforms (many SVR4 systems) there is no way to force the linker to export all global symbols for use in DSO's when linking the Apache httpd executable program. But without the visibility of the Apache core symbols no standard Apache module could be used as a DSO. The only workaround here is to use the SHARED_CORE feature because this way the global symbols are forced to be exported. As a consequence the Apache src/Configure script automatically enforces SHARED_CORE on these platforms when DSO features are used in the Configuration file or on the configure command line.

1.15 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- Stas Bekman [http://stason.org/]

1.16 Authors

- Stas Bekman [http://stason.org/]

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: