CHAPTER 2

Getting Started Fast

This chapter is about getting started with mod_perl, for the very impatient. If all you want is to run your existing CGI scripts in a mod_perl-enabled environment, we'll try to make this as easy for you as possible. Of course, we hope that you'll read the rest of the book too. But first, we want to show you how simple it is to harness the power of mod_perl.

On a decent machine, it should take half an hour or less to compile and configure a mod_perl-based Apache server and get it running. Although there are binary distributions of mod_perl-enabled Apache servers available for various platforms, we recommend that you always build mod_perl from source. It's simple to do (provided you have all the proper tools on your machine), and building from source circumvents possible problems with binary distributions, such as those reported for the RPM packages built for Red Hat Linux.

The mod_perl installation that follows has been tested on many mainstream Unix and Linux platforms. Unless you're using a very nonstandard system, you should have no problems when building the basic mod_perl server.

For Windows users, the simplest solution is to use the binary package available from *http://perl.apache.org/download/binaries.html*. Windows users may skip to the section entitled "Installing mod_perl for Windows."

Before we continue, however, we have one important bit of advice: while you're learning mod_perl, be sure that you experiment on a private machine and not on a production server.

Installing mod_perl 1.0 in Three Steps

You can install mod_perl in three easy steps: obtain the source files required to build mod_perl, build mod_perl, and install it.

Building mod_perl from source requires a machine with basic development tools. In particular, you will need an ANSI-compliant C compiler (such as *gcc*) and the *make*

,ch02.21082 Page 26 Thursday, November 18, 2004 12:34 PM

utility. All standard Unix-like distributions include these tools. If a required tool is not already installed, you can install it with the package manager that is provided with the system (*rpm*, *apt*, *yast*, etc.).

A recent version of Perl (5.004 or higher) is also required. Perl is available as an installable package, although most Unix-like distributions will have Perl installed by default. To check that the tools are available and to learn about their version numbers, try:

```
panic% make -v
panic% gcc -v
panic% perl -v
```

If any of these responds with `Command not found`, the utility will need to be installed.

Once all the tools are in place, the installation can begin. Experienced Unix users will need no explanation of the commands that follow and can simply type them into a terminal window.

Get the source code distrubutions of Apache and mod_perl using your favorite web browser or a command-line client such as *wget* or *lwp-download*. These two distributions are available from *http://www.apache.org/dist/httpd/* and *http://perl.apache.org/dist/*, respectively.

The two packages are named *apache_1.3.xx.tar.gz* and *mod_perl-1.xx.tar.gz*, where *1.3.xx* and *1.xx* should be replaced with the real version numbers of Apache and mod_perl, respectively. Although 2.0 development versions of Apache and mod_perl are available, this book covers the mod_perl 1.0 and Apache 1.3 generation, which were the stable versions when this book was written. See Chapters 24 and 25 for more information on the Apache 2.0 and mod_perl 2.0 generation.

Move the downloaded packages into a directory of your choice (for example, */home/stas/src/*), proceed with the following steps, and mod_perl will be installed:

```
panic% cd /home/stas/src
panic% tar -zvxf apache_1.3.xx.tar.gz
panic% tar -zvxf mod_perl-1.xx.tar.gz
panic% cd mod_perl-1.xx
panic% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \
    APACHE_PREFIX=/home/httpd DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
panic% make && make test
panic% su
panic# make install
```

All that remains is to add a few configuration lines to the Apache configuration file (*/usr/local/apache/conf/httpd.conf*), start the server, and enjoy mod_perl.

Installing mod_perl on Unix Platforms

Now let's go over the installation again, this time with each step explained in detail and with some troubleshooting advice. If the build worked and you are in a hurry to boot your new *httpd*, you may skip to the section entitled “Installing mod_perl for Windows.”

Before installing Apache and mod_perl, you usually have to become *root* so that the files can be installed in a protected area. However, users without *root* access can still install all files under their home directories by building Apache in an unprivileged location; you need *root* access only to install it. We will talk about the nuances of this approach in Chapter 3.

Obtaining and Unpacking the Source Code

The first step is to obtain the source code distributions of Apache and mod_perl. These distributions can be retrieved from *http://www.apache.org/dist/httpd/* and *http://perl.apache.org/dist/* and are also available from mirror sites. Even if you have the Apache server running on your machine, you’ll need its source distribution to rebuild it from scratch with mod_perl.

The source distributions of Apache and mod_perl should be downloaded into a directory of your choice. For the sake of consistency, we assume throughout the book that all builds are being done in the */home/stas/src* directory. Just remember to substitute */home/stas/src* in the examples with the actual path being used.

The next step is to move to the directory containing the source archives:

```
panic% cd /home/stas/src
```

Uncompress and un*tar* both sources. GNU *tar* allows this using a single command per file:

```
panic% tar -zvxf apache_1.3.xx.tar.gz
panic% tar -zvxf mod_perl-1.xx.tar.gz
```

For non-GNU *tar*s, you may need to do this with two steps (which you can combine via a pipe):

```
panic% gzip -dc apache_1.3.xx.tar.gz | tar -xvf -
panic% gzip -dc mod_perl-1.xx.tar.gz | tar -xvf -
```

Linux distributions supply *tar* and *gzip* and install them by default. If your machine doesn’t have these utilities already installed, you can get *tar* and *gzip* from *http://www.gnu.org/*, among other sources. The GNU versions are available for every platform that Apache supports.

Building mod_perl

Move into the */home/stas/src/mod_perl-1.xx/* source distribution directory:

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

The next step is to create the *Makefile*. This is no different in principle from the creation of the *Makefile* for any other Perl module.

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

mod_perl accepts a variety of parameters. The options specified above will enable almost every feature that mod_perl offers. There are many other options for fine-tuning mod_perl to suit particular circumstances; these are explained in detail in Chapter 3.

Running *Makefile.PL* will cause Perl to check for prerequisites and identify any required software packages that are missing. If it reports missing Perl packages, they will have to be installed before proceeding. Perl modules are available from CPAN (*http://cpan.org/*) and can easily be downloaded and installed.

An advantage of installing mod_perl with the help of the `CPAN.pm` module is that all the missing modules will be installed with the `Bundle::Apache` bundle:

```
panic% perl -MCPAN -e 'install("Bundle::Apache")'
```

We will talk in depth about using `CPAN.pm` in Chapter 3.

Running *Makefile.PL* also transparently executes the *./configure* script from Apache's source distribution directory, which prepares the Apache build configuration files. If parameters must be passed to Apache's *./configure* script, they can be passed as options to *Makefile.PL*. Chapter 3 covers all this in detail.

The *httpd* executable can now be built by using the *make* utility (note that the current working directory is still */home/stas/src/mod_perl-1.xx/*):

```
panic% make
```

This command prepares the mod_perl extension files, installs them in the Apache source tree, and builds the *httpd* executable (the web server itself) by compiling all the required files. Upon completion of the *make* process, the working directory is restored to */home/stas/src/mod_perl-1.xx/*.

Running *make test* will execute various mod_perl tests on the newly built *httpd* executable:

```
panic% make test
```

This command starts the server on a nonstandard port (8529) and tests whether all parts of the built server function correctly. The process will report anything that does not work properly.

Installing mod_perl

Running *make install* completes the installation process by installing all the Perl files required for mod_perl to run. It also installs the mod_perl documentation (manpages). Typically, you need to be *root* to have permission to do this, but another user account can be used if the appropriate options are set on the *perl Makefile.PL* command line (see Chapter 3). To become *root*, use the *su* command.

```
panic% su
panic# make install
```

If you have the proper permissions, you can also chain all three *make* commands into a single command line:

```
panic# make && make test && make install
```

The single-line version simplifies the installation, since there is no need to wait for each command to complete before starting the next one. Of course, if you need to become *root* in order to run *make install*, you'll either need to run *make install* as a separate command or become *root* before running the single-line version.

If you choose the all-in-one approach and any of the *make* commands fail, execution will stop at that point. For example, if *make* alone fails, then *make test* and *make install* will not be attempted. Similarly, if *make test* fails, then *make install* will not be attempted.

Finally, change to the Apache source distribution directory and run *make install* to create the Apache directory tree and install Apache's header files (**.h*), default configuration files (**.conf*), the *httpd* executable, and a few other programs:

```
panic# cd ../apache_1.3.xx
panic# make install
```

Note that, as with a plain Apache installation, any configuration files left from a previous installation will not be overwritten by this process. Although backing up is never unwise, it's not actually necessary to back up the previously working configuration files before the installation.

At the end of the *make install* process, the installation program will list the path to the *apachectl* utility, which you can use to start and stop the server, and the path to the installed configuration files. It is important to write down these pathnames, as they will be needed frequently when maintaining and configuring Apache. On our machines, these two important paths are:

```
/usr/local/apache/bin/apachectl
/usr/local/apache/conf/httpd.conf
```

The mod_perl Apache server is now built and installed. All that needs to be done before it can be run is to edit the configuration file *httpd.conf* and write a test script.

Configuring and Starting the mod_perl Server

Once you have mod_perl installed, you need to configure the server and test it.

The first thing to do is ensure that Apache was built correctly and that it can serve plain HTML files. This helps to minimize the number of possible problem areas:

once you have confirmed that Apache can serve plain HTML files, you know that any problems with mod_perl are related to mod_perl itself.

Apache should be configured just as you would configure it without mod_perl. Use the defaults as suggested, customizing only when necessary. Values that will probably need to be customized are `ServerName`, `Port`, `User`, `Group`, `ServerAdmin`, `DocumentRoot`, and a few others. There are helpful hints preceding each directive in the configuration files themselves, with further information in Apache's documentation. Follow the advice in the files and documentation if in doubt.

When the configuration file has been edited, start the server. One of the ways to start and stop the server is to use the *apachectl* utility. To start the server with *apachectl*, type:

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

To stop the server, type:

```
panic# /usr/local/apache/bin/apachectl stop
```

Note that if the server will listen on port `80` or another privileged port,* the user executing *apachectl* must be *root*.

After the server has started, check in the *error_log* file (*/usr/local/apache/logs/error_log*, by default) to see if the server has indeed started. Do not rely on the *apachectl* status reports. The *error_log* should contain something like the following:

```
[Thu Jun 22 17:14:07 2000] [notice] Apache/1.3.12 (Unix)
mod_perl/1.24 configured -- resuming normal operations
```

Now point your browser to *http://localhost/* or *http://example.com/*, as configured with the `ServerName` directive. If the `Port` directive has been set with a value other than `80`, add this port number to the end of the server name. For example, if the port is 8080, test the server with *http://localhost:8080/* or *http://example.com:8080/*. The "It Worked!" page, which is an *index.html* file that is installed automatically when running *make install* in the Apache source tree, should appear in the browser. If this page does not appear, something went wrong and the contents of the *logs/error_log* file should be checked. The path to the error log file is specified by the `ErrorLog` directive in *httpd.conf*. (It is usually specified relative to the `ServerRoot`, so a value of *logs/error_log* usually means */usr/local/apache/logs/error_log* if Apache is installed into */usr/local/apache*.)

If everything works as expected, shut down the server, open *httpd.conf* with a text editor, and scroll to the end of the file. The mod_perl configuration directives are conventionally added to the end of *httpd.conf*. It is possible to place mod_perl's configuration directives anywhere in *httpd.conf*, but adding them at the end seems to work best in practice.

* Privileged ports are 0–1023. Only the programs running as *root* are allowed to bind to these.

Assuming that all the scripts that should be executed by the mod_perl-enabled server are located in the */home/stas/modperl* directory, add the following configuration directives:

```
Alias /perl/ /home/stas/modperl/

PerlModule Apache::Registry
<Location /perl/>
    SetHandler perl-script
    PerlHandler Apache::Registry
    Options +ExecCGI
    PerlSendHeader On
    Allow from all
</Location>
```

Save the modified file.

This configuration causes every URI starting with */perl* to be handled by the Apache mod_perl module with the handler from the Perl module `Apache::Registry`.

Installing mod_perl for Windows

Apache runs on many flavors of Unix and Unix-like operating systems. Version 1.3 introduced a port to the Windows family of operating systems, often named Win32 after the name of the common API. Because of the many differences between Unix and Windows, the Win32 port of Apache is still branded as beta quality—it hasn't yet reached the stability and performance levels of the native Unix counterpart.

Another hindrance to using mod_perl on Windows is that current versions of Perl are not thread-safe on Win32. As a consequence, mod_perl calls to the embedded Perl interpreter must be serialized (i.e., executed one at a time). For these reasons, we recommend that mod_perl on Windows be used only for testing purposes, not in production.

Building mod_perl from source on Windows is a bit of a challenge. Development tools such as a C compiler are not bundled with the operating system, and most users expect a point-and-click installation, as with most Windows software. Additionally, all software packages need to be built with the same compiler and compile options. This means building Perl, Apache, and mod_perl from source, which is quite a daunting task.

Fortunately, Randy Kobes maintains a Windows distribution of mod_perl that includes all the necessary tools, including Perl, Apache, and a host of useful CPAN modules. Using this distribution provides an out-of-the-box Apache + mod_perl combo in minutes.

The distribution comes with extensive documentation. Take the time to read it, particularly if you want to install the software in a location different from the default. In the following installation, we'll use the default locations and options.

Here are the steps for installing mod_perl:

1. Download the Windows distribution. Download *perl-win32-bin-x.x.exe* from *http://perl.apache.org/download/binaries.html*. This self-extracting archive yields four directories: *Apache/*, *Perl/*, *openssl/*, and *readmes/*.
2. Install the software. Move the *Apache/* and *Perl/* directories to *C:\*. Edit *C:\AUTOEXEC.BAT* to install the Perl executable directories in your system's search path:

   ```
   SET PATH=C:\Perl\5.6.1\bin;C:\Perl\5.6.1\bin\MSWin32-x86;"%PATH%"
   ```

   Then restart Windows for this change to take effect.
3. Test the Perl installation. Open a DOS prompt window to verify that Perl is installed correctly and learn the version number:

   ```
   C:\> perl -v

   This is perl, v5.6.1 built for MSWin32-x86

   Copyright 1987-2000, Larry Wall
   ```
4. Start Apache. The distribution comes with a ready-made configuration file for mod_perl, which we'll use to start Apache. From the *C:\Apache* directory, start Apache:

   ```
   C:\Apache> apache.exe -f conf\httpd.conf
   ```

   Now, issuing a request for *http://localhost/* displays the usual Apache "It Worked!" page.
5. Test mod_perl. The distribution comes with a preconfigured mod_perl handler and `Apache::Registry` directory. We can test our mod_perl-enabled server by issuing the following requests:

   ```
   http://localhost/hello
   http://localhost/mod_perl/printenv
   ```

We now have a fully functional mod_perl server. The example scripts described in the rest of this chapter can be used with minor modifications to file paths and URIs. In particular, change all instances of */home/stas* to *C:\Apache\*, and change all instances of *http://localhost/perl* to *http://localhost/mod_perl*.

Installing mod_perl with the Perl Package Manager

If you are already a Perl developer on Windows, it is likely that you have ActivePerl (see *http://www.activestate.com/*) installed. In that case, you can get a mod_perl distribution that takes advantage of your existing Perl installation.

First of all, you will need to get the latest Apache distribution. Go to *http://www.apache.org/dist/httpd/binaries/win32/* and get the latest version of *apache_1.3.xx-win32-no_src.msi*, which is a graphical installer. Read the notes on that page about the MSI Binary distribution carefully if you are using Windows NT 4.0 or Windows 9x, as there may be some prerequisites.

There is a lot of documentation at *http://httpd.apache.org/* about installing Apache on Windows, so we won't repeat it here. But for the purposes of this example, let's suppose that your Apache directory is *C:\Apache*, which means you chose *C:\* as the installation directory during the installation of Apache, and it created a subdirectory named *Apache* there.

Once Apache is installed, we can install mod_perl. mod_perl is distributed as a *PPM* file, which is the format used by the ActivePerl *ppm* command-line utility. mod_perl isn't available from ActiveState, but it has been made available from a separate archive, maintained by Randy Kobes.* To install mod_perl, do the following from a DOS prompt:

```
C:\> ppm
PPM> install mod_perl
PPM> quit
C:\>
```

When *install mod_perl* completes, a post-installation script will run, asking you where to install *mod_perl.so*, the mod_perl dynamic link library (DLL) that's used by Apache. Look over the suggested path and correct it if necessary, or press Enter if it's correct; it should be the *C:\Apache\modules* directory if you used *C:\Apache* as an installation directory.

Please note that the version of mod_perl provided in that archive is always the latest version of mod_perl compiled against the latest version of Apache, so you will need to make sure you have the latest Apache (of the *1.3.x* series) installed before proceeding. Furthermore, you will need an ActivePerl installation from the 6xx series, based on Perl 5.6.x, or mod_perl won't work.

The next step is to enable mod_perl in your *httpd.conf* file. If you installed Apache in *C:\Apache*, this will be *C:\Apache\conf\httpd.conf*.

Add this line together with any other `LoadModule` directives:

```
LoadModule perl_module modules/mod_perl.so
```

Furthermore, if you have a `ClearModuleList` directive in the same file, add the following line with the other `AddModule` directives:

```
AddModule mod_perl.c
```

For more information, see the Apache documentation for these two directives, and see Chapter 3 for more information on using mod_perl as a dynamic shared object (DSO).

With this installation, you can start Apache as described in its documentation, and try out the examples in this book. However, the mod_perl test scripts cited above

* See the Preface for more information about PPM installation.

aren't provided, and you will have to configure mod_perl yourself. See Chapter 4 for more information about configuring mod_perl. For example:

```
Alias /perl/ C:/Apache/perl/

PerlModule Apache::Registry
<Location /perl/>
    SetHandler perl-script
    PerlHandler Apache::Registry
    Options +ExecCGI
    PerlSendHeader On
    Allow from all
</Location>
```

This will allow you to run `Apache::Registry` scripts placed in the directory *C:\Apache\perl*. As you may have noticed, we use forward slashes instead of the backslashes that are used on Windows (i.e., *C:/Apache/perl/* instead of *C:\Apache\perl\*), to be compatible with Unix notation.

Preparing the Scripts Directory

Now you have to select a directory where all the mod_perl scripts and modules will be placed. We usually create a directory called *modperl* under our home directory for this purpose (e.g., */home/stas/modperl*), but it is also common to create a directory called *perl* under your Apache server root, such as */usr/local/apache/perl*.

First create this directory if it doesn't yet exist:

```
panic% mkdir /home/stas/modperl
```

Next, set the file permissions. Remember that when scripts are executed from a shell, they are being executed with the permissions of the user's account. Usually, you want to have read, write, and execute access for yourself, but only read and execute permissions for the server. When the scripts are run by Apache, however, the server needs to be able to read and execute them. Apache runs under an account specified by the `User` directive, typically *nobody*. You can modify the `User` directive to run the server under your username, for example:

```
User stas
```

Since the permissions on all files and directories should usually be `rwx------`,[*] set the directory permissions to:

```
panic% chmod 0700 /home/stas/modperl
```

Now no one but you and the server can access the files in this directory. You should set the same permissions for all the files you place under this directory. [†]

[*] See the `chmod` manpage for more information regarding octal modes.

[†] You don't need to set the `x` bit for files that aren't going to be executed; mode `0600` is sufficient for those files.

If the server is running under the *nobody* account, you have to set the permissions to `rwxr-xr-x` or `0755` for your files and directories. This is insecure, because other users on the same machine can read your files.

```
panic# chmod 0755  /home/stas/modperl
```

If you aren't running the server with your username, you have to set these permissions for all the files created under this directory so Apache can read and execute them.

In the following examples, we assume that you run the server under your username, and hence we set the scripts' permissions to `0700`.

A Sample Apache::Registry Script

One of mod_perl's benefits is that it can run existing CGI scripts written in Perl that were previously used under mod_cgi (the standard Apache CGI handler). Indeed, mod_perl can be used for running CGI scripts without taking advantage of any of mod_perl's special features, while getting the benefit of the potentially huge performance boost. Example 2-1 gives an example of a very simple CGI-style mod_perl script.

Example 2-1. mod_perl_rules1.pl

```
print "Content-type: text/plain\n\n";
print "mod_perl rules!\n";
```

Save this script in the */home/stas/modperl/mod_perl_rules1.pl* file. Notice that the `#!` line (colloquially known as the *shebang* line) is not needed with mod_perl, although having one causes no problems, as can be seen in Example 2-2.

Example 2-2. mod_perl_rules1.pl with shebang line

```
#!/usr/bin/perl
print "Content-type: text/plain\n\n";
print "mod_perl rules!\n";
```

Now make the script executable and readable by the server, as explained in the previous section:

```
panic% chmod 0700 /home/stas/modperl/mod_perl_rules1.pl
```

The *mod_perl_rules1.pl* script can be tested from the command line, since it is essentially a regular Perl script:

```
panic% perl /home/stas/modperl/mod_perl_rules1.pl
```

This should produce the following output:

```
Content-type: text/plain

mod_perl rules!
```

Make sure the server is running and issue these requests using a browser:

```
http://localhost/perl/mod_perl_rules1.pl
```

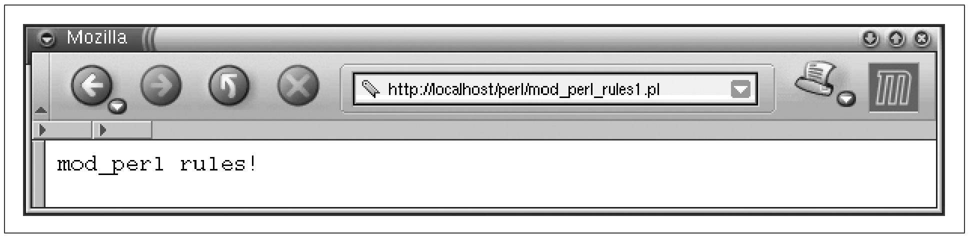
If the port being used is not 80 (e.g., 8080), the port number should be included in the URL:

```
http://localhost:8080/perl/mod_perl_rules1.pl
```

Also, the `localhost` approach will work only if the browser is running on the same machine as the server. If not, use the real server name for this test. For example:

```
http://example.com/perl/mod_perl_rules1.pl
```

The page rendered should be similar to the one in Figure 2-1.



```
mod_perl rules!
```

Figure 2-1. Testing the newly configured server

If you see it, congratulations! You have a working mod_perl server.

If something went wrong, go through the installation process again, making sure that none of the steps are missed and that each is completed successfully. You might also look at the *error_log* file for error messages. If this does not solve the problem, Chapter 3 will attempt to salvage the situation.

Jumping a little bit ahead, Example 2-3 shows the same CGI script written with the mod_perl API.

Example 2-3. mod_perl_rules2.pl

```
my $r = Apache->request;
$r->send_http_header('text/plain');
$r->print("mod_perl rules!\n");
```

The mod_perl API needs a request object, `$r`, to communicate with Apache. The script retrieves this object and uses it to send the HTTP header and print the irrefutable fact about mod_perl's coolness.

This script generates the same output as the previous one.

As you can see, it's not much harder to write your code using the mod_perl API. You need to learn the API, but the concepts are the same. As we will show in the following chapters, usually you will want to use the mod_perl API for better performance or when you need functionality that CGI doesn't provide.

Porting Existing CGI Scripts to mod_perl

Now it's time to move any existing CGI scripts from the */somewhere/cgi-bin* directory to */home/stas/modperl*. Once moved, they should run much faster when requested from the newly configured base URL (*/perl/*). For example, a CGI script called *test.pl* that was previously accessed as */cgi-bin/test.pl* can now be accessed as */perl/test.pl* under mod_perl and the `Apache::Registry` module.

Some of the scripts might not work immediately and may require some minor tweaking or even a partial rewrite to work properly with mod_perl. We will talk in depth about these issues in Chapter 6. Most scripts that have been written with care and developed with warnings enabled and the `strict` pragma[*] will probably work without any modifications at all.

A quick solution that avoids most rewriting or editing of existing scripts that do not run properly under `Apache::Registry` is to run them under `Apache::PerlRun`. This can be achieved by simply replacing `Apache::Registry` with `Apache::PerlRun` in *httpd.conf*. Put the following configuration directives instead in *httpd.conf* and restart the server:

```
Alias /perl/ /home/stas/modperl/
PerlModule Apache::PerlRun
<Location /perl/>
    SetHandler perl-script
    PerlHandler Apache::PerlRun
    Options ExecCGI
    PerlSendHeader On
    Allow from all
</Location>
```

Almost every script should now run without problems; the few exceptions will almost certainly be due to the few minor limitations that mod_perl or its handlers have, but these are all solvable and covered in Chapter 6.

As we saw in Chapter 1, `Apache::PerlRun` is usually useful while transitioning scripts to run properly under `Apache::Registry`. However, we don't recommend using `Apache::PerlRun` in the long term; although it is significantly faster than mod_cgi, it's still not as fast as `Apache::Registry` and mod_perl handlers.

A Simple mod_perl Content Handler

As we mentioned in the beginning of this chapter, mod_perl lets you run both scripts and handlers. The previous example showed a script, which is probably the most familiar approach to web programming, but the more advanced use of mod_perl

[*] Warnings and `strict` abort your script if you have written sloppy code, so that you won't be surprised by unknown, hidden bugs. Using them is generally considered a good thing in Perl and is *very* important in mod_perl.

involves writing handlers. Have no fear; writing handlers is almost as easy as writing scripts and offers a level of access to Apache's internals that is simply not possible with conventional CGI scripts.

To create a mod_perl handler module, all that is necessary is to wrap the code that would have been the body of a script into a `handler` subroutine, add a statement to return the status to the server when the subroutine has successfully completed, and add a package declaration at the top of the code.

Just as with scripts, the familiar CGI API may be used. Example 2-4 shows an example.

Example 2-4. ModPerl/Rules1.pm

```
package ModPerl::Rules1;
use Apache::Constants qw(:common);

sub handler {
    print "Content-type: text/plain\n\n";
    print "mod_perl rules!\n";
    return OK; # We must return a status to mod_perl
}
1; # This is a perl module so we must return true to perl
```

Alternatively, the mod_perl API can be used. This API provides almost complete access to the Apache core. In the simple example used here, either approach is fine, but when lower-level access to Apache is required, the mod_perl API shown in Example 2-5 must be used.

Example 2-5. ModPerl/Rules2.pm

```
package ModPerl::Rules2;
use Apache::Constants qw(:common);

sub handler {
    my $r = shift;
    $r->send_http_header('text/plain');
    $r->print("mod_perl rules!\n");
    return OK; # We must return a status to mod_perl
}
1; # This is a perl module so we must return true to perl
```

Create a directory called *ModPerl* under one of the directories in `@INC` (e.g., under */usr/lib/perl5/site_perl/5.6.1*), and put *Rules1.pm* and *Rules2.pm* into it. (Note that you will need *root* access in order to do this.) The files should include the code from the above examples. To find out what the `@INC` directories are, execute:

```
panic% perl -le 'print join "\n", @INC'
```

On our machine it reports:

```
/usr/lib/perl5/5.6.1/i386-linux
/usr/lib/perl5/5.6.1
```

```
/usr/lib/perl5/site_perl/5.6.1/i386-linux
/usr/lib/perl5/site_perl/5.6.1
/usr/lib/perl5/site_perl
.
```

Therefore, on our machine, we might place the files in the directory */usr/lib/perl5/site_perl/5.6.1/ModPerl*. By default, when you work as *root*, the files are created with permissions allowing everybody to read them, so here we don't have to adjust the file permissions (the server only needs to be able to read those).

Now add the following snippet to */usr/local/apache/conf/httpd.conf*, to configure mod_perl to execute the `ModPerl::Rules1::handler` subroutine whenever a request to *mod_perl_rules1* is made:

```
PerlModule ModPerl::Rules1
<Location /mod_perl_rules1>
   SetHandler perl-script
   PerlHandler ModPerl::Rules1
   PerlSendHeader On
</Location>
```

Now issue a request to:

```
http://localhost/mod_perl_rules1
```

and, just as with the *mod_perl_rules.pl* scripts, the following should be rendered as a response:

```
mod_perl rules!
```

Don't forget to include the port number if not using port 80 (e.g., *http://localhost:8080/mod_perl_rules1*); from now on, we will assume you know this.

To test the second module, `ModPerl::Rules2`, add a similar configuration, while replacing all `1`s with `2`s:

```
PerlModule ModPerl::Rules2
<Location /mod_perl_rules2>
   SetHandler perl-script
   PerlHandler ModPerl::Rules2
</Location>
```

In Chapter 4 we will explain why the `PerlSendHeader` directive is not needed for this particular module.

To test, use the URI:

```
http://localhost/mod_perl_rules2
```

You should see the same response from the server that we saw when issuing a request for the former mod_perl handler.

,ch02.21082 Page 40 Thursday, November 18, 2004 12:34 PM

Is This All We Need to Know About mod_perl?

So do you need to know more about mod_perl? The answer is, “Yes and no.”

Just as with Perl, effective scripts can be written even with very little mod_perl knowledge. With the basic unoptimized setup presented in this chapter, visitor counters and guestbooks and any other CGI scripts you use will run much faster and amaze your friends and colleagues, usually without your changing a single line of code.

However, although a 50 times improvement in guestbook response times is great, a very heavy service with thousands of concurrent users will suffer under a delay of even a few milliseconds. You might lose a customer, or even many of them.

When testing a single script with the developer as the only user, squeezing yet another millisecond from the response time seems unimportant. But it becomes a real issue when these milliseconds add up at the production site, with hundreds or thousands of users concurrently generating requests to various scripts on the site. Users are not merciful nowadays. If there is another site that provides the same kind of service significantly faster, chances are that users will switch to the competing site.

Testing scripts on an unloaded machine can be very misleading—everything might seem so perfect. But when they are moved into a production environment, chances are that the scripts will not behave as well as they did on the development box. For example, the production machine may run out of memory on very busy services. In Chapter 10, we will explain how to optimize code to use less memory and how to make as much memory as possible shared.

Debugging is something that some developers prefer not to think about, because the process can be very tedious. Learning how to make the debugging process simpler and more efficient is essential for web programmers. This task can be difficult enough when debugging CGI scripts, but it can be even more complicated with mod_perl. Chapter 21 explains how to approach debugging in the mod_perl environment.

mod_perl has many features unavailable under mod_cgi for working with databases. Some of the most important are persistent database connections. Persistent database connections require a slightly different approach, explained in Chapter 20.

Most web services, especially those aimed at an international audience, must run nonstop, 24×7. But at the same time, new scripts may need to be added and old ones removed, and the server software will need upgrades and security fixes. And if the server goes down, fast recovery is essential. These issues are considered in Chapter 5.

Finally, the most important aspect of mod_perl is the mod_perl API, which allows intervention at any or every stage of request processing. This provides incredible flexibility, allowing the creation of scripts and processes that would simply be impossible with mod_cgi.

There are many more things to learn about mod_perl and web programming in general. The rest of this book will attempt to provide as much information as possible about these and other related matters.

References

- The Apache home page: *http://www.apache.org/*.
- The mod_perl home page: *http://perl.apache.org/*.
- The CPAN home page: *http://cpan.org/*

  CPAN is the Comprehensive Perl Archive Network. Its aim is to contain all the Perl material you will need. The archive is close to a gigabyte in size at the time of this writing, and CPAN is mirrored at more than 100 sites around the world.
- The `libwww-perl` home page: *http://www.linpro.no/lwp/*.

  The `libwww-perl` distribution is a collection of Perl modules and programs that provide a simple and consistent programming interface (API) to the World Wide Web. The main focus of the library is to provide classes and functions that facilitate writing WWW clients; thus, `libwww-perl` is said to be a WWW client library. The library also contains modules that are of more general use, as well as some useful programs.