Apache

Brief introduction in Apache Web Server

Prepared by Chris Panayiotou for
EPL 602

What is Apache?

- A public-domain open source Web server
  - developed by a loosely-knit group of programmers
  - core development of the Apache Web server is performed by a group of volunteer programmers, called the *Apache Group*
- The first version of Apache, based on the NCSA httpd Web server, was developed in 1995
- Because the source code is freely available, anyone can adapt the server for specific needs
  - there is a large public library of Apache add-ons
  - similar to development of the Linux operating system
- The original version of Apache was written for UNIX
  - now there are versions that run under OS/2, Windows and other platforms
- The name is a tribute to the Native American Apache Indian tribe, a tribe well known for its inexhaustible endurance and skill in warfare

What is Apache?

- is a powerful, flexible, HTTP/1.1 compliant web server
- implements the latest protocols, including HTTP/1.1 (RFC2616)
- is highly configurable and extensible with third-party modules
- can be customized by writing 'modules' using the Apache module API
- provides full source code and comes with an unrestrictive license
- runs on Windows NT/9x, Netware 5.x and above, OS/2, and most versions of Unix, as well as several other operating systems
- is actively being developed
- encourages user feedback through new ideas, bug reports and patches

Apache's Features

- implements many frequently requested features, including:
  - DBM databases for authentication
    - ✓ allows to easily set up password-protected pages with enormous numbers of authorized users, without bogging down the server
  - Customized responses to errors and problems
    - ✓ allows to set up files, or even CGI scripts, which are returned by the server in response to errors and problems
    - ✓ e.g. setup a script to intercept 500 Server Errors and perform on-the-fly diagnostics
  - Multiple DirectoryIndex directives
    - ✓ allows you to say DirectoryIndex index.html index.cgi, which instructs the server to either send back index.html or run index.cgi when a directory URL is requested, whichever it finds in the directory
  - Unlimited flexible URL rewriting and aliasing
    - ✓ Apache has no fixed limit on the numbers of Aliases and Redirects which may be declared in the config files.
    - ✓ in addition, a powerful rewriting engine can be used to solve most URL manipulation problems.

Apache's Features

- Content negotiation
  - ✓ automatically serve clients of varying sophistication and HTML level compliance, with documents which offer the best representation of information that the client is capable of accepting.
- Virtual Hosts
  - ✓ distinguish between requests made to different IP addresses or names (mapped to the same machine). Apache also offers dynamically configurable mass-virtual hosting.
- Configurable Reliable Piped Logs
  - ✓ generate logs in the format that you want.
  - ✓ on most Unix architectures, can send log files to a pipe, allowing for log rotation, hit filtering, real-time splitting of multiple vhosts into separate logs, and asynchronous DNS resolving on the fly.

Why Apache?

- Notable for playing a key role in the initial growth of the WWW
- Apache has been the most popular web server on the Internet since April of 1996
  - continues to be the most popular web server in use
  - the November 2006 WWW server site survey by Netcraft found that 60.3% of the web sites on the Internet are using Apache
    - ✓ more widely used than all other web servers combined
- The *de facto* reference platform against which other web servers are designed and judged
- The goal of this project is to provide a secure, efficient and extensible server which provides HTTP services in sync with the current HTTP standards
- It has a free license and it's public domain, open source

Basic Configuration

- Generally, for the home hobbyist, there is no need to do any editing at all to apache's configuration file
- You can locate the config file usually by looking for the file **/etc/httpd/conf/httpd.conf**
- The config file is broken up into three sections
  - the Global Section
  - the Main (or default server) section
  - and the Virtual Hosts section.
  - In older versions of Apache, two additional files, **srm.conf** and **access.conf** controlled resources and access rights.
    - ✓ still kept around, but are now deprecated.

Section 1: Global Section

- This section controls behavior that is global to all instances of apache running on your system
- The example configuration file contains excellent documentation for each of the options

Directive	**Hints**
ServerRoot	If you configured sysconfdir to be /etc/httpd/conf then make this "/etc/httpd"
LockFile	This file is used by apache to decide if it's running or not. If the path does not start with "/" will assume path is relative to the **ServerRoot**.
pidfile	This file is where apache stores the process id of the server. If the path does not start with "/" will assume path is relative to the **ServerRoot**.
ScoreBoardFile	This file stores internal server information, but is not needed on most Linux configurations. Just to be safe, create a place for it.
TimeOut	This is the number of seconds before net traffic times out. The default on this is 300. Values below 30 tend to cause problems.
KeepAlive	Allows persistent connections. Unless you have a good reasons to not want them, set this to "on".
MaxKeepAliveRequests	This determines the maximum number of Requests allowed on a persistent channel before it closes. 100 is a reasonable number.

Section 1: Global Section

Directive	Hints
KeepAliveTimeout	Determines how long a KeepAlive channel will remain open if idle. 15 is a good number.
MinSpareServers	Sets the desired number of servers that are idle, awaiting requests. If there are ever less than this many of idle child processes, apache will start spawning more until this number is reached. Too many wastes resources. Too few and spikes in server hits could degrade performance. 2 is a good number for home or SOHO, 3 - 5 for a business or small university.
MaxSpareServers	Sets the maximum desired number of idle servers. If there are more idle servers than desired, apache will begin to kill off children, reclaiming their resources. 10 is the default, while for the hobbyist or SOHO user, a value of 5 can be used to save resources.
StartServers	The number of children to spawn at startup. The default is 5. Busy sites should set this higher, but not too high or you'll spend your first minute and a half spawning children and not serving requests. Apache will dynamically adjust the number of processes later, so setting this value very high is almost never useful.
MaxClients	This sets a ceiling on the number of child processes that can be spawned. It can be set up to 256 without modifying source code.

Section 1: Global Section

Directive	Hints
MaxRequestsPerChild	This sets the maximum number of requests that a child process will handle before dying. It is mainly useful on IRIX and SunOS where there are noticeable memory leaks in the libraries. A value of 0 will allow unlimited requests per child, and is claimed to be safe on Linux. I recommend a value of 1000, or 10000 for heavily loaded sites.
Listen	Determines the address and port number that apache will bind. This can be used to limit apache to a specific address. For instance, you can use Listen 127.0.0.1:80 to cause apache to respond only to requests from the localhost. The usual value is 80, which tells apache to listen on the HTTP port of all interfaces. Multiple Listen directives can be used.
BindAddress	Determines which IP addresses apache will respond to. This is used on machines with multiple IP addresses (either through multiplexing or using multiple interfaces). The normal value is *, which causes apache to listen on all addresses.
ExtendedStatus	This is only useful if you have loaded mod_status, and tells apache to keep track of extended information on a per request basis. It cannot be used on a virtualhost by virtualhost basis. Set this value to "on" if you've decided to compile mod_status as a built-in module (recommended).

Section 1: Global Section

Directive	Hints
ClearModuleList	Apache has a list of modules that should be active. This directive clears that list. It is assumed that you will then turn on what you want using the AddModule directive.
AddModule	Modules are sort of complicated. When you compile apache, it gets a list of included modules, not all of which are "turned on". This directive is used to activate a built-in module. It can be used even if you haven't used the ClearModuleList directive.
LoadModule	This directive is used to load a dynamically loaded module (as opposed to a built-in module. Order of execution can be important, so pay close attention to the example configuration and the documentation for any alternative modules you load.
<IfDefine></IfDefine>	This is used to conditionally execute directives based on whether or not a specific value is defined, usually by means of a command line switch (-D foo). One use for this is for a startup script to check for the existence of a module, and load/configure it if it exists.

Section 2: Main (Default Server) Section

- Deals with the default server
  - The default server (or main server) is the one that will handle any requests not captured by a <VirtualHost>
- Directives and instructions that you set in this section are, in general, inherited by virtualhosts as well
  - can set some good default behaviors here rather than duplicating a lot of effort
  - settings inside <VirtualHost> will override these options for that particular virtualhost

Directive	Hints
Port	Here for historical reasons, and for setting the SERVER_PORT environment variable for CGI and SSI. Set this to whatever your HTTP port will be (usually 80). Note: This does NOT apply to virtualhosts.
User	Sets the user that apache will handle requests as. For security reasons, apache changes its effective UID before handling requests, so all of your documents must be accessible to this user. For this reason, it is useful to create a user called *www* or *apache* to use with your webserver. Running as the user *nobody* or as UID -1 does not work on all systems or with all libraries.
Group	Just as apache changes its UID, it also changes its GID. This is the group to change to. Once again, nobody can cause you some difficult to track-down problems, so it's probably a good idea to create a group.
ServerAdmin	Set this to the e-mail address that should receive all error notifications.
ServerName	Set this to the fully qualified domain name of the server. Also used when setting up name-based virtual hosts. If you don't, you will likely encounter problems on startup.

Section 2: Main Section

Directive	Hints
DocumentRoot	Set this to the directory to search for the main index file for this server. Apache will search for a file that matches your DirectoryIndex in this directory to display when no other page is requested
UserDir	When using the mod_userdir module, this allows you to map requests to user's home directories instead of to the document root tree. Set this to "www" to map requests for *http://example.org/~foo* to *~foo/www* on the example.org server, for example. **For security reasons, if you use this, also use UserDir Disabled root.**
DirectoryIndex	Used with mod_dir, this option sets the search order for files when a user requests a directory listing by specifying a "/" at the end of a directory name or for the document root. Normally this will just return "index.html", but you could specify DirectoryIndex index.html index.php index.pl index.cgi to have apache search for each of these files, returning the first one it found.
HostNameLookups	Generally set to "off" to save the latency time of the DNS lookup, you can set this to either "on" or "double". "On" is useful to pass the hostname as REMOTE_HOST to CGI/SSI's and "Double" is the ultra-paranoid setting to detect spoofed requests. On heavily loaded sites this can cause some real slowdown, and most poeple don't need it.
ErrorLog	Sets the name of the file to use for error logging. As of version 1.3, you can also direct errors to the syslog facility.

Section 2: Main Section

Directive	Hints
LogLevel	Sets the level of information that apache will send to the error log. Defaults to "error". Possible options are "emerg", "alert", "crit", "error", "warn", "notice", "info", and "debug"
LogFormat	When using mod_log_config (recommended), this directive allows you to customize the format of the log file. The options are many and various. Read the documentation. The most commonly used is LogFormat "%h %l %u %t \"%r\" %>s %b" for main host, and LogFormat "%v %l %u %t \"%r\" %>s %b" for virtual hosts.
Alias	Allows for transparent redirection of requests. Typically used for icon, library image, and cgi directory redirection on a wholesale basis. Aliases are processed after <Location> stanzas and before <Directory> stanzas.
ScriptAlias	Has the same result as Alias, but also marks the directory as containing cgi scripts, so apache will process them as such.
AddHandler	If using mod_mime (recommended) this directive maps file extensions to handlers. An example of this is using AddHandler cgi-script .cgi to cause any file with the extension .cgi to be treated as a cgi file. This overrides any previous mappings.

Section 2: Main Section

Directive	Hints
AddType	If using mod_mime (recommended) this directive maps file extensions to MIME types. One particularly forward looking use for this directive is mapping the ".xhtml" extension to text/html. An example of this is using `AddType text/html .xhtml` to cause any file with the extension .xhtml to be treated as html by the client. Converting your html to xhtml will generally only have small impacts on presentation, which can almost always be mediated with proper adjustments to CSS. While it isn't fully desirable to treat xhtml as html, no major browser is fully XHTML aware as of yet.
ErrorDocument	Allows you to set custom pages or scripts to handle HTTP exceptions and errors. This lets you get away from the canned error messages and allows for a more friendly and effective way to handle things like broken links and access denial. Example: `ErrorDocument 404 errordocs/404.cgi` would invoke a custom error script when a file is not found on the server (bad typing or broken/obsolete link).

Section 3: Virtual Servers

- Virtual servers are a way for a single invocation of apache to serve multiple domain names.
- Three ways to go about it:
  - port based
    - ✓ commonly used to serve HTTP and HTTPS from the same server
  - address based
    - ✓ used primarily for backward compatibility to HTTP 1.0 clients, which don't transmit the desired hostname as part of the request.
  - named based
    - ✓ the most commonly used method
    - ✓ multiple domain names share the same IP address (CNAME aliasing)
      - commonly used by web hosting services to preserve IP space
      - SOHO's who wish to serve something like *www.my_business.com* and *www.my_personal_page.net* from the same server
    - ✓ cannot be used with SSL secure servers because of the way the SSL protocol works.

Section 3: Virtual Servers

- The third section of the apache configuration file deals with virtual servers
  - Virtual servers are defined in a <VirtualHost> *stanza*
    - ✓ Stanzas are almost like HTML tags
      - they start with a *<keyword>* in angle braces, and end with *</keyword>*
    - ✓ Directives inside stanzas only apply within the scope defined by that stanza. E.g.:
      - **<Directory /home/foouser/public_html/*>**
        Order Deny, Allow
        Deny from Joe
        Allow from All
        </Directory>
        User Joe would have no access to files located under /home/foouser/public_html, but his access would remain unaffected for all other areas

Setting up name based virtual hosts

- Assume that www.example.com and www.foo.org point to the same IP address
- In **httpd.conf** file add the following:

```
NameVirtualHost *
<VirtualHost>
    ServerAdmin webmaster@example.com
    DocumentRoot /www/docs/example.com
    ServerName example.com
    ErrorLog logs/example.com_error
</VirtualHost>
<VirtualHost>
    ServerAdmin webmaster@foo.org
    DocumentRoot /www/docs/foo.org
    ServerName foo.org
    ErrorLog logs/foo.org_error
</VirtualHost>
```

- You may want to enable or disable certain features for each virtual host
  - Simply place the appropriate directives in the virtual hosts stanza

Setting up name based virtual hosts

- What if you want to host hundreds of virtual hosts?
  - httpd.conf would grow huge, be slow to load, and consume a lot of resources
- Dynamically configured mass virtual hosting provided by mod_vhost_alias
- If you enable this module, either as a dynamic module or built-in, you can use something like this:

```
# Turn off Canonical Names so CGI/SSI works properly
UseCanonicalName off

# Set the logging format for all virtual hosts
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# Dynamically include server names in file requests
VirtualDocumentRoot /www/vhosts/%0/htdocs
VirtualScriptAlias /www/vhosts/%0/cgi-bin
```

- With this setup, a request to http://www.virtualhost.com/foo/bar.html would map to /www/vhosts/www.virtualhost.com/htdocs/foo/bar.html
  - You can still use <Directory> and other stanzas to control things on a directory by directory basis.

Dynamic Content

- Dynamic content includes things like negotiated content CGI, PHP, Perl generated pages, and SSI (Server Side Includes)
- **Negotiated Content**
  - Beginning with HTTP 1.1, browsers have been able to send information to the server specifying additional information and preferences
    - ✓ The browser can inform the web server that it will accept GIF images, but would really prefer PNG or JPEG if they're available
  - Apache can parse these preferences and react to them
    - ✓ The common request headers that Apache understands are Accept, Accept-Language, Accept-Charset, and Accept-Encoding.
    - ✓ Apache's negotiation rules can be quite complex, but basic negotiation is actually quite easy
      - First, ensure that mod_negotiation is enabled for your server
      - Second, add a handler for type-map, usually by including the configuration directive **AddHandler type-map .var**
      - Third set up the type-map files themselves. Then instead of hyper-linking to an image file or web-page, you hyperlink to the .var file

Example of Negotiated Content

- Create a file called foo.var, and create a hyperlink to it. Fill in the contents like this:

```
URI: foo.english.html
Content-type: text/html
Content-language: en

URI: foo.french.html
Content-type: text/html
Content-language: fr

URI: foo.german.html
Content-type: text/html
Content-language: de
```

- Now when the user cliks on the link, Apache looks the Accept-language header and will return the right file.

Example of Negotiated Content

- You can do the same thing with images. If you had a link like `<IMG SRC=./foo.var>` and the foo.var file contained

```
URI: foo.jpeg
Content-type: image/jpeg; qs=0.8

URI: foo.gif
Content-type: image/gif; qs=0.5

URI: foo.png
Content-type: image/png; qs=0.3
```

- Apache would look for the **Accept-encoding** header in the request, and return the type of image that was
  - in the list of acceptable encodings,
  - had the highest qs value (these range from 1.000 to 0.000)
- Now lets say you have a case where *none* options in your .var file are acceptable to the browser.
  - Apache will return error 406 (NOT ACCEPTABLE), and a hyperlinked list of the possible options.
  - cool feature with translated pages, but tends not to work too well with images

Transparent Content Negotiation

- Apache offers what is called "transparent content negotition"
  - Enable **Multiviews** in the Options directive
  - Have files like foo.en.html, foo.fr.html, foo.de.html, and foo.html,
  - Simply hyper-link to "foo", with no extension
- Apache will fake up a type-map on the fly, and serve the best match
  - Good idea to have a "default", like foo.html which, since it has no encoding or language specified at all, is always acceptable to the browser
- You could "simply" use **mod_actions** to re-write documents into the desired format on the fly using CGI scripts
  - Need a really fast server
  - Lots of time to write the translators

CGI

- CGI refers to the Common Gateway Interface,
  - is the most common method of executing external programs or scripts on the server side to generate content
- Even things like PHP make use of the concepts of CGI to perform their functions and features
- CGI can also be your worst security nightmare
  - Instructions on enabling CGI in Apache can be found in the CGI HOWTO included with the Apache documentation
  - Be aware that the default setting for the **Options** directive is "All", which allows executing CGI's from anywhere they are found.
    - ✓ This can be a big security hole in and of itself

Getting CGI to work

- **Needed Modules:** mod_alias, mod_cgi, mod_mime
- **Configuration Directives:** AddHandler, Options, ScriptAlias
- Add to your configuration file:
  - AddModule mod_mime.c, AddModule mod_cgi.c, AddModule mod_alias.c
  - ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
    - ✓ maps requests for **http://www.example.com/cgi-bin/foo** to the script **/home/httpd/cgi-bin/foo**
    - ✓ tells Apache that every file in the cgi-bin directory should be treated as a CGI script
  - AddHandler cgi-script cgi
    - ✓ tells apache that files that ends with .cgi should be treated as a CGI program;
    - ✓ this example will work anywhere in the document tree, not just the cgi-bin directory
    - ✓ you only need this if you wish execution of CGI's outside the ScriptAlias'ed directory.
  - You could drop this directive into < VirtualHost> or <Directory> stanzas to limit its scope.

```
Options –ExecCGI
<Directory /foo/bar/ >
          Options +ExecCGI
<Directory>
<Directory /home/httpd/*/www/cgi-bin/ >
          Options +ExecCGI
<Directory>
```

    - ✓ Disables CGI exection globally , but allows it for the /foo/bar directory and any directory with a name that matches /home/httpd/*/www/cgi-bin.
  - Interaction between **ScriptAlias**, **Options**, and the **AddHandler** directives can be tricky

PHP

- While compiling and installing PHP as a module for your Apache webserver is a bit tricky, it is well worth the effort.
  - Luckily most distributions come with PHP already
  - If you're compiling your own PHP, download the latest stable source from the PHP homepage, and unpack it
  - There are about a hundred configuration options, many with their own particular dependencies, so configuring the source tree can be very difficult
    - ✓ to get exactly the features you want, it's the only way to go
  - You'll need the apache source tree as well,
    - ✓ if you want to build your own PHP, you'll almost have to build your own apache as well
- Configuring Apache for PHP: Simply add the following lines to your httpd.conf file:

```
# Use the next line if PHP is a DSO, omit it otherwise
LoadModule php4_module /path/to/php3/module/libphp4.so

# These lines need to go in for both DSO and static
AddModule mod_php4.c
AddType application/x-httpd-php4 .php4 .php
```

Perl and mod_perl

- Perl, while not being written from the ground up for web-use like PHP was, has an enormous existing code-base.
- With the advent of mod_perl's server-embedded Perl engine, its now fairly fast to not only use Perl scripts as CGI's, but to actually code **entire Apache extension modules in Perl**
- Compilation and installation of mod_perl is similar to compiling and installing PHP
- Configuring Apache for mod_perl
  - There are lot of ways to configure Apache with mod_perl
  - In fact, using PERL directives, it's completely possible to re-write *httpd.conf* completely in perl!
  - For basic functionality just add the following:

```
# for Apache::Registry Mode
Alias /perl/ "/home/httpd/cgi-bin/"
# for Apache::Perlrun Mode
Alias /cgi-perl/ "/home/httpd/cgi-bin/“

# For /perl/* as apache modules written in perl
<Location /perl>
  Perlrequire /path/to/apache/modules/perl/startup.perl
  PerlModule Apache::Registry
  SetHandler perl-script
  PerlHandler Apache::Registry
  Options ExecCGI
  PerlSendHeader On
</Location>
```

Configuring Apache for mod_perl

```
# For /cgi-perl/* handling as embedded perl
 <Location /cgi-perl>
   SetHandler perl-script
   PerlHardler Apache::PerlRun
   Options ExecCGI
   PerlSendHeader On
 </Location>

 # For mod_perl status information
 <Location /perl-status>
   SetHandler perl-script
   PerlHandler Apache::Status
   order deny, allow
   deny from all
   allow from localhsot
 </Location>

 # Include the next line if mod_perl is a DSO
   LoadModule perl_module /path/to/apache/modules/libperl.so

 AddModule mod_perl.c
```

- There is plenty of additional information available both in the pod files that come with mod_perl, the apache module help file, and on the mod_perl home page

Server Side Includes (SSI)

- Much like html pages with embedded scripts, SSI is just another set of what can be thought of as almost HTML tags
- SSI allows for an easy way to include right in the middle of a web page such things as file modification time, values of environment variables, current date and time, and even the output of programs and scripts
- It differs from standard CGI in that the "included" information is parsed right into an html file, rather than the entire content being generated by a program
- The apache documentation carries a quite good tutorial
- Configuring Apache for SSI: configure and compile mod_include (either as DSO or static), and add a few lines to the config file:

```
# Use this to allow SSI in files. This can go in stanzas, too.
Options +Includes
# Or you can have SSI but disable executing scripts via SSI with
Options +IncludesNOEXEC

# Use this if mod_include is a DSO
LoadModule includes_module /path/to/apache/modules/mod_include.so

AddModule mod_include.c
AddType text/html .shtml
AddHandler server-parsed .shtml

# Optionally, you could run *all* html files through the SSI parser.
# This does no harm to non SSI html files, but slows you down a bit
AddHandler server-parsed .html
```

Basic Concepts

- Apache breaks down request handling into a series of steps. These are:
  - *URI* → Filename translation
  - *Auth ID* checking [is the user who they say they are?]
  - *Auth access* checking [is the user authorized *here*?]
  - *Access* checking other than auth
  - Determining *MIME type* of the object requested
  - *"Fixups"*
  - Actually *sending a response* back to the client.
  - *Logging* the request
- These phases are handled by looking at each of a succession of *modules*, looking to see if each of them has a handler for the phase, and attempting invoking it if so. The handler can typically do one of three things:
  - *Handle* the request, and indicate so by returning the magic constant OK
  - *Decline* to handle the request, by returning the magic integer constant DECLINED.
    - ✓ The server behaves as if the handler simply hadn't been there
  - Signal an error, by returning one of the HTTP error codes
    - ✓ terminates normal handling of the request
    - ✓ an ErrorDocument may be invoked
    - ✓ it will be logged in any case.

Basic Concepts

- Most phases are terminated by the first module that handles them
  - for logging, "fixups", and non-access authentication checking, all handlers always run (barring an error)
  - The response phase is unique in that modules may declare multiple handlers for it, via a dispatch table keyed on the MIME type of the requested object
  - Modules may declare a response-phase handler which can handle *any* request, by giving it the key */* (*i.e.*, a wildcard MIME type specification)
    - ✓ Wildcard handlers are only invoked if the server has already tried and failed to find a more specific response handler for the MIME type of the requested object
      - either none existed, or they all declined
    - ✓ The handlers themselves are functions of one argument (a pointer to request_rec structure), which returns an integer.

Basic Concepts

- Include needed Apache libraries
  - An Apache module will require information about structures, macros and functions from Apache's core
  - `#include "httpd.h"`
    `#include "http_config.h"`
    - ✓ These two header files are the most basic, but real modules will need to include other header files relating to request handling, logging, protocols, etc. E.g:
      - `#include "http_core.h"`
        `#include "http_log.h"`
        `#include "http_protocol.h"`
- Register your module in Apache
  - Notify Apache which phases of the request your module handles
  - Notify Apache for which content type it provides handler
- Write the respective handlers
- Compile and integrate it with Apache
  - Different procedures for each platform

Hello World Module

```
/* File: mod_hello.c */

/* Apache libraries */
#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_protocol.h"

/* Make the name of the content handler known to Apache */
static handler_rec hello_handlers[] =
{
   {"hello-handler", hello_handler},
   /* Could also add handlers for a specific MIME type e.g.:
    *    {“application/x-httpd-app”, handle_app},
    * Then you would need to define the handler function “handle_app”
    * that produced a response for requests of that MIME type.
    */
   {NULL}
};
```

Hello World Module

```
/* Tell Apache what phases of the transaction we handle */
module MODULE_VAR_EXPORT hello_module =
{
  STANDARD_MODULE_STUFF,
  NULL,                /* module initializer                  */
  NULL,                /* per-directory config creator        */
  NULL,                /* dir config merger                   */
  NULL,                /* server config creator               */
  NULL,                /* server config merger                */
  NULL,                /* command table                       */
  hello_handlers,      /* [9]  content handlers               */
  NULL,                /* [2]  URI-to-filename translation    */
  NULL,                /* [5]  check/validate user_id         */
  NULL,                /* [6]  check user_id is valid *here*  */
  NULL,                /* [4]  check access by host address   */
  NULL,                /* [7]  MIME type checker/setter       */
  NULL,                /* [8]  fixups                         */
  NULL,                /* [10] logger                         */
  NULL,                /* [3]  header parser                  */
  NULL,                /* process initialization              */
  NULL,                /* process exit/cleanup                */
  NULL                 /* [1]  post read_request handling     */
};
```

Hello World Module

```
/* The content handler */
static int hello_handler(request_rec *r) {
 const char* hostname;
  r->content_type = "text/html";
 ap_send_http_header(r);
 hostname = ap_get_remote_host(r->connection,r->per_dir_config,REMOTE_NAME);
 ap_rputs("<HTML>\n"                              ,r);
 ap_rputs("<HEAD>\n"                              ,r);
 ap_rputs("<TITLE>Hello There</TITLE>\n"          ,r);
 ap_rputs("</HEAD>\n"                             ,r);
 ap_rputs("<BODY>\n"                              ,r);
 ap_rprintf(r,"<H1>Hello %s</H1>\n"               ,hostname);
 ap_rputs(“Would you take this seriously if the first example didn't\n",r);
 ap_rputs("say \"hello world\"?\n"                ,r);
 ap_rputs("</BODY>\n"                             ,r);
 ap_rputs("</HTML>\n"                             ,r);
  return OK;
}
```

Adding Hello World Module to Apache

- Compile it either as
  - Static Apache Library
    - ✓ From the top of the Apache distribution directory type this command:
      ```
      % ./configure --activate-module=src/modules/site/mod_hello.c
                    --enable-module=hello
      ```
    - ✓ A new make is now available. All you have to do is to recompile Apache!
  - Dynamic/Shared Library
    - ✓ From the top of the Apache distribution run the *configure* command
      ```
      % ./configure --activate-module=src/modules/site/mod_hello.c
                    --enable-shared=hello
      ```
    - ✓ Now you'll need to run *make* to create the file *src/modules/site/mod_hello.so*. When this is done, just copy the shared object file to Apache's *libexec* directory
    - ✓ Add the following lines to *httpd.conf*:
      ```
      LoadModule hello_module modules/mod_hello.so
      ```
- Invoking the module
  - Add the following lines to *httpd.conf*:
    ```
    <Location /hi/there>
       SetHandler hello-handler
    </Location>
    ```