

In this chapter:

- Apache Program Architecture
- Apache Kernel Functionality
- Apache Module Functionality

Chapter 2

Apache Functionality



*Good design means less design.
Design must serve users,
not try to fool them.*

— Dieter Rams,
Chief Designer, Braun

Apache is a very complex web server, mainly because of the vast number of features provided. Fortunately, most of this functionality stays in clearly separated and independent program modules, which facilitates program understanding and maintenance. In this chapter, we look at the Apache program architecture, consisting mainly of a program kernel and various optional modules. We then introduce each module by describing its purpose and the directives that it implements. The order in which modules are presented in this chapter will be repeated again in the other chapters. You can therefore treat this chapter as an overview of the Apache program as a whole and as a departure point from which to examine particular functionalities and implemented directives.

2.1 Apache Architecture

Figure 2.1 on the next page depicts Apache's program architecture. This layering architecture consists of four layers, which are built on top of one another.

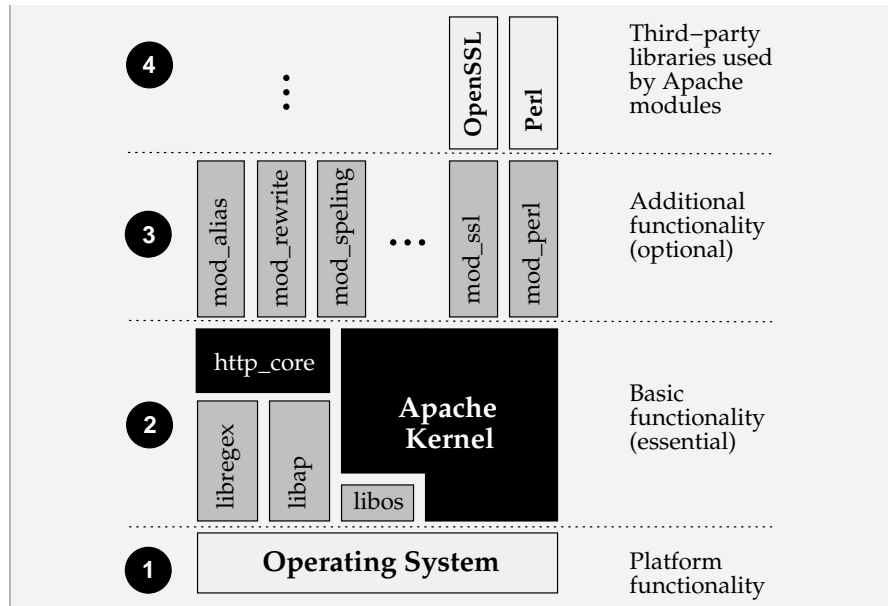


Figure 2.1: The architecture of the Apache web server

❶ Operating System

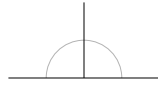
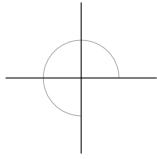
The basic functionality is provided by the underlying operating system. For Apache, this operating system is typically some flavor of UNIX (p.14), but it can also be Win32, OS/2, MacOS, or even the POSIX subsystems of a mainframe.

❷ Apache Kernel, Core Module, and Kernel Libraries

Layer 2 is the main Apache program, consisting of an Apache kernel, a core module, and a few standard libraries. The Apache kernel, together with the special core module (`http_core`), implements the basic HTTP server functionality and provides the Apache application programming interface (API) to the module layer. This layer also contains a library of generic, reusable code (`libap`), a library that implements regular expression parsing and matching (`libregex`), and a small operating system abstraction library (`libos`).

❸ Apache Modules

The impressive user-visible functionality that makes Apache unique among the existing web servers is provided by lots of Apache modules on layer 3. Usually each module implements one clearly separated functionality. In reality, no module is required. Running a minimal web server capable only of serving static documents from a configured



document area is possible even without any modules.¹ This chapter focuses on the standard modules of the Apache program distribution.

④ Third-Party Libraries

For the standard modules (those found in the official Apache distribution), this layer is usually empty.² Additional modules, such as `mod_ssl` and `mod_perl`, use external third-party libraries, however; these libraries can be found on this layer of the Apache architecture.

The interesting part of this program architecture is the fact that layers 3 and 4 are loosely coupled with layer 2; whereas all modules on layer 3 are designed to remain independent of one another.³ A side effect of this architecture is that the program code of layers 3 and 4 cannot be statically linked with the program code of layer 1.

In combination with the Dynamic Shared Object (DSO) facility, this structure provides great flexibility. One can therefore assemble the Apache functionality provided by layers 3 and 4 at start-up time (instead of at installation time!) by letting the Apache kernel load the necessary parts.⁴

2.2 Apache Kernel Functionality

The Apache kernel (layer 2 in Figure 2.1 on the facing page) has two purposes: (1) to provide the basic HTTP functionality, and (2) to provide the module API.

■ Basic HTTP Server Functionality

The kernel must support resource handling (through file descriptors, memory segments, and so on), maintain the pre-forked process model, listen to the TCP/IP sockets of the configured virtual servers, transfer control of incoming HTTP requests to the handler processes, handle the HTTP protocol states, and provide read/write buffers, among other duties. Additionally, it provides general functionality like URL and MIME header parsing, DSO loading, and many more capabilities.

■ Apache Module API

As already mentioned, the real functionality of Apache resides inside modules. To allow these modules to fully control the Apache processing, the kernel must provide an API. In Apache, this API consists of a

¹In practice, one at least requires `mod_mime`.

²There might be some exceptions. For instance, some modules need a NDBM library that must usually be provided as an external library when it is not part of the vendor's C library.

³Technically, they are not totally independent of one another, because of ordering issues and the shared process address space.

⁴Technically speaking, `mod_so` loads the DSOs and not the kernel.



static function list in each module (which the kernel uses to dispatch messages between the modules while processing a HTTP request) and a set of API functions (all starting with the common prefix “ap_”) that the modules can use. Each HTTP request is divided into ten distinct steps, and each module can hook into each step. At each step, a module can usually either decline or accept to handle the step. To handle the step, the module calls back the kernel through various `ap_xxx()` functions.

For more details about the internals of the Apache API, refer to both the comprehensive documentation inside *Writing Apache Modules with Perl and C* (Lincoln Stein and Doug MacEachern, O’Reilly & Associates Inc., 1999) and the online API documentation under <http://dev.apache.org/apidoc/>.

2.3 Apache Module Functionality

The real user-visible functionality of Apache resides in the various Apache modules. Currently (as of Apache 1.3), the Apache program distribution comes with the core module plus 36 additional standard modules. In this section, we introduce all of these modules plus two important third-party modules: `mod_ssl` and `mod_perl`. Many more third-party modules exist, of course. Each addresses specialized problem situations and solutions. This book, however, covers only the most important modules.

When you need additional functionality, first search for a solution in the *Apache Module Registry* (<http://modules.apache.org/>). The chance is high that you will find a solution there, as more than 140 modules have been registered.

2.3.1 Core Functionality

■ `http_core` (enabled by default)

Apache Base Functionality

Since Apache 1.0, `src/main/http_core.c`

The Apache Group (1994)

`http_core` is the base module of Apache, in which all core functionality is implemented. Although this module also uses the Apache Module API, it is a special one: it has a nonstandard file name (`http_core` instead of the expected `mod_core`), it works with special non-API links, and links between the Apache internals and this module are mandatory. In other words, although you can usually strip down Apache at buildtime by removing unnecessary modules, the `http_core` module can never be removed.

Although the core module `http_core` uses the Apache Module API, it is not a regular module, because it has hard-coded links and dealings with the kernel.

Directives:

<code></Directory></code> (→ p.71)	<code>KeepAliveTimeout</code> (→ p.81)
<code></DirectoryMatch></code> (→ p.71)	<code>LimitRequestBody</code> (→ p.81)
<code></Files></code> (→ p.72)	<code>LimitRequestFields</code> (→ p.81)
<code></FilesMatch></code> (→ p.72)	<code>LimitRequestFieldsize</code> (→ p.82)
<code></IfDefine></code> (→ p.74)	<code>LimitRequestLine</code> (→ p.82)
<code></IfModule></code> (→ p.74)	<code>Listen</code> (→ p.83)
<code></Limit></code> (→ p.73)	<code>ListenBacklog</code> (→ p.83)
<code></Location></code> (→ p.70)	<code>LockFile</code> (→ p.84)
<code></LocationMatch></code> (→ p.70)	<code>LogLevel</code> (→ p.84)
<code></VirtualHost></code> (→ p.69)	<code>MaxClients</code> (→ p.84)
<code><Directory></code> (→ p.71)	<code>MaxKeepAliveRequests</code> (→ p.85)
<code><DirectoryMatch></code> (→ p.71)	<code>MaxRequestsPerChild</code> (→ p.85)
<code><Files></code> (→ p.72)	<code>MaxSpareServers</code> (→ p.85)
<code><FilesMatch></code> (→ p.72)	<code>MinSpareServers</code> (→ p.85)
<code><IfDefine></code> (→ p.74)	<code>NameVirtualHost</code> (→ p.86)
<code><IfModule></code> (→ p.74)	<code>Options</code> (→ p.86)
<code><Limit></code> (→ p.73)	<code>PidFile</code> (→ p.87)
<code><Location></code> (→ p.70)	<code>Port</code> (→ p.87)
<code><LocationMatch></code> (→ p.70)	<code>RLimitCPU</code> (→ p.87)
<code><VirtualHost></code> (→ p.69)	<code>RLimitMEM</code> (→ p.88)
<code>AccessConfig</code> (→ p.74)	<code>RLimitNPROC</code> (→ p.88)
<code>AccessFileName</code> (→ p.75)	<code>Require</code> (→ p.89)
<code>AddModule</code> (→ p.75)	<code>ResourceConfig</code> (→ p.89)
<code>AllowOverride</code> (→ p.75)	<code>Satisfy</code> (→ p.89)
<code>AuthName</code> (→ p.76)	<code>ScoreBoardFile</code> (→ p.90)
<code>AuthType</code> (→ p.76)	<code>SendBufferSize</code> (→ p.90)
<code>BindAddress</code> (→ p.77)	<code>ServerAdmin</code> (→ p.90)
<code>ClearModuleList</code> (→ p.77)	<code>ServerAlias</code> (→ p.91)
<code>ContentDigest</code> (→ p.77)	<code>ServerName</code> (→ p.91)
<code>CoreDumpDirectory</code> (→ p.78)	<code>ServerPath</code> (→ p.91)
<code>DefaultType</code> (→ p.78)	<code>ServerRoot</code> (→ p.91)
<code>DocumentRoot</code> (→ p.78)	<code>ServerSignature</code> (→ p.92)
<code>ErrorDocument</code> (→ p.78)	<code>ServerTokens</code> (→ p.92)
<code>ErrorLog</code> (→ p.79)	<code>ServerType</code> (→ p.92)
<code>Group</code> (→ p.79)	<code>StartServers</code> (→ p.93)
<code>HostnameLookups</code> (→ p.80)	<code>Timeout</code> (→ p.93)
<code>IdentityCheck</code> (→ p.80)	<code>UseCanonicalName</code> (→ p.93)
<code>Include</code> (→ p.80)	<code>User</code> (→ p.94)
<code>KeepAlive</code> (→ p.81)	

■ **mod_so** (disabled by default)

Dynamic Shared Object (DSO) Bootstrapping

Since Apache 1.2, `src/modules/standard/mod_so.c`

Robert S. Thau, Alexei Kosut, Paul Sutton, Ralf S. Engelschall (1996)

`mod_so` is a very interesting module. It supports the DSO facility, which Apache provides for building modules as stand-alone units (shared object files) and loading them at runtime into the address space of the `httpd` process. Thus, although `mod_so` is implemented as a module, it provides some bootstrapping functionality for other modules that one usually would expect to find inside `http_core`.

`mod_so` allows you to load other modules on demand.

Directives:

LoadFile (→ p.94) **LoadModule** (→ p.95)

2.3.2 URL Mapping

■ **mod_alias** (enabled by default)

Simple URL Translation and Redirection

Since Apache 1.0, `src/modules/standard/mod_alias.c`

Rob McCool, David Robinson, Robert S. Thau (1995)

`mod_alias` is the father of all URL manipulation modules. It has existed since the early Apache days and provides a limited, but easy-to-understand mechanism for mapping URLs to file names. The original idea was that one could translate URLs to file names by mapping URL prefixes to directory paths on the file system. The `AliasMatch` and `RedirectMatch` directives use regular expressions instead of prefixes to achieve more flexibility.

`mod_alias` is intended for standard URL-to-file name mappings.

Directives:

Alias (→ p.95)	RedirectPermanent (→ p.97)
AliasMatch (→ p.96)	RedirectTemp (→ p.97)
Redirect (→ p.96)	ScriptAlias (→ p.97)
RedirectMatch (→ p.96)	ScriptAliasMatch (→ p.97)

■ **mod_rewrite** (disabled by default)

Advanced URL Translation and Redirection

Since Apache 1.2, `src/modules/standard/mod_rewrite.c`

Ralf S. Engelschall (1996)

`mod_rewrite` is the Swiss Army Knife of URL manipulations. It provides virtually all of the functions one would ever need to manipulate URLs, and its functionality is highly generalized. Consequently, `mod_rewrite` can be used to solve all sorts of URL-based problems. The drawback is the high learning curve, because this module is based on a complex rule-based matching engine, which uses regular expressions for its patterns. Although the flexibility of `mod_rewrite` makes it a very complex tool, once you understand the basic idea you will master all existing and forthcoming URL-based problems in your webmaster's life.

`mod_rewrite` is the most powerful URL manipulation solution.

Directives:

RewriteBase (→ p.100)	RewriteLogLevel (→ p.99)
RewriteCond (→ p.100)	RewriteMap (→ p.99)
RewriteEngine (→ p.98)	RewriteOptions (→ p.98)
RewriteLock (→ p.99)	RewriteRule (→ p.101)
RewriteLog (→ p.98)	

■ **mod_userdir** (enabled by default)

URL Selection by User Names

Since Apache 1.0, `src/modules/standard/mod_userdir.c`

Rob McCool, Alexei Kosut, Ken Coar (1995)

`mod_userdir` is a module specialized in mapping (`/~username`) URLs to the home pages of the corresponding users; these home pages are usually found inside the home directory of the user or under one or more dedicated home page areas.

`mod_userdir` finds the home pages of your users.

Directive:

UserDir (→ p.102)

■ **mod_imap** (enabled by default)

URL Selection by Image Map Coordinates

Since Apache 1.0, `src/modules/standard/mod_imap.c`

Rob McCool, Kevin Hughes, Randy Terbush, James H. Cloos, Jr., Nathan Kurz, Mark Cox (1995)

The task of `mod_imap` is simply to determine the surrounding area of `x,y`-coordinates (given in the `QUERY_STRING`) inside a server-based *image map* and perform an HTTP redirection to the URL corresponding to this area. Although this task is a very specialized one, do not underestimate the difficulty involved in solving this problem. Because image maps can contain arbitrarily complex polygons, a dedicated module to handle this task is only reasonable.

`mod_imap` maps image map coordinates to URLs.

Directives:

ImapBase (→ p.102) **ImapMenu** (→ p.103)

ImapDefault (→ p.103)

■ **mod_speling** (disabled by default)

URL Spelling Correction

Since Apache 1.3, `src/modules/standard/mod_speling.c`

Alexei Kosut, Martin Kraemer (1997)

`mod_speling` is a very handy module. It corrects minor spelling or capitalization errors in URLs — indeed its droll name provides a hint as to its task. The module addresses this problem by trying to find a matching document, even after all other modules (such as `mod_alias`, `mod_rewrite`, or `mod_userdir`) give up. It works by comparing each document name in the requested directory against the requested document name without regard to case, allowing a maximum of one misspelling (character insertion, omission, transposition, or wrong character). The drawback of this nice feature is that the complicated disk I/O usually increases the response time. Often, it is a better choice to force the user to fix the reference.

`mod_speling` is a nifty URL spell checker that corrects document URLs on-the-fly.

Directive:

CheckSpelling (→ p.104)

2.3.3 Access Control

■ **mod_access** (enabled by default)

Host- and Network-Based Access Control

Since Apache 1.0, `src/modules/standard/mod_access.c`

Rob McCool (1995)

`mod_access`, as its name clearly implies, provides access control for documents. It allows one to restrict or allow access to resources based on the client's host name, IP address, or network address. This module serves as Apache's basic access control mechanism.

`mod_access` restricts access through network identifiers.

Directives:

Allow (→ p.104) **Order** (→ p.105)

Deny (→ p.105)

2.3.4 User Authentication

■ **mod_auth** (enabled by default)

User Authentication by User Name/Password

Since Apache 1.0, `src/modules/standard/mod_auth.c`

Rob McCool, Robert S. Thau, Dirk van Gulik (1995)

The authentication module `mod_auth` deals with the HTTP Basic Authentication facility, which is simply a user name/password pair submitted by the client together with the request for a document. This module allows one to check this information against a flat-file database similar to UNIX's `/etc/passwd` and `/etc/group` files and to deny access when the given user name/password doesn't match the database information. Special variants of this module exist that offer the same functionality but use a database from other than a flat-file (for performance reasons).

`mod_auth` provides the HTTP Basic Authentication facility.

Directives:

AuthAuthoritative (→ p.106) **AuthUserFile** (→ p.106)

AuthGroupFile (→ p.106)

■ **mod_auth_anon** (disabled by default)

User Authentication by Anonymous Name/E-Mail Address

Since Apache 1.1, `src/modules/standard/mod_auth_anon.c`

Rob McCool, Brian Behlendorf, Robert S. Thau, Dirk van Gulik (1996)

The functionality of `mod_auth_anon` resembles the behavior of an Anonymous-FTP server, in some ways. That is, this module deals with the Basic Authentication facility like `mod_auth`. On the other hand, it does not serve any real access control purposes. Instead, it merely identifies the user. One can therefore give the `REMOTE_USER` variable a meaning in SSI/CGI scripts or log files (for noncritical distinguishing purposes) without requiring real user authentication.

`mod_auth_anon` resembles the Anonymous-FTP idea.

Directives:

Anonymous (→ p.107)	Anonymous_MustGiveEmail (→ p.108)
Anonymous_Authoritative (→ p.107)	Anonymous_NoUserId (→ p.108)
Anonymous_LogEmail (→ p.107)	Anonymous_VerifyEmail (→ p.108)

■ `mod_auth_dbm` (disabled by default)

User Authentication by User Name/Password (UNIX NDBM)

Since Apache 1.0, `src/modules/standard/mod_auth_dbm.c`

Rob McCool, Robert S. Thau, Dirk van Gulik (1996)

`mod_auth_dbm` is a variant of `mod_auth` that provides exactly the same functionality, but uses a standard UNIX NDBM hash file instead of a flat-file database. The advantage is a magnitude-better performance in the lookups (performed for every request) — an especially important consideration when the user community is very large. An NDBM library is provided by most all UNIX platforms.

`mod_auth_dbm` is the NDBM-based variant of `mod_auth`.

Directives:

AuthDBMAuthoritative (→ p.108)	AuthDBMUserFile (→ p.109)
AuthDBMGroupFile (→ p.109)	

■ `mod_auth_db` (disabled by default)

User Authentication by User Name/Password (Berkeley-DB)

Since Apache 1.1, `src/modules/standard/mod_auth_db.c`

Rob McCool, Brian Behlendorf, Robert S. Thau, Andrew Cohen (1996)

`mod_auth_db` is another variant of `mod_auth` that provides exactly the same functionality. Instead of a flat-file database, however, it uses a Berkeley-DB/1.x or Berkeley-DB/2.x hash file. The advantage is a magnitude-better performance in the lookups (performed for every request) — an especially important consideration when the user community is very large. The Berkeley-DB library is usually not provided by UNIX platform vendors, but is more reliable and faster than NDBM libraries.

`mod_auth_db` is the Berkeley-DB-based variant of `mod_auth`.

Directives:

AuthDBAuthoritative (→ p.110)	AuthDBUserFile (→ p.111)
AuthDBGroupFile (→ p.110)	

■ **mod_digest** (disabled by default)

User Authentication by User Name/Realm/Password

Since Apache 1.1, `src/modules/standard/mod_digest.c`

Rob McCool, Robert S. Thau, Alexei Kosut (1996)

In addition to the classical HTTP/1.0 Basic Authentication mechanism, a message digest-based HTTP authentication mechanism exists as defined in RFC 2617.⁵ Instead of transferring a clear-text user name/password pair with the HTTP request (which can be easily monitored), a message digest is calculated (via the MD5 algorithm) and transferred together with the user name. This module then performs the same message digest calculation for the password stored in the server's authentication database. When the two digests are equal, access is allowed. This approach offers an obvious advantage relative to Basic Authentication: the password is not sent over the network. The drawback is that many browsers do not support this type of user authentication.

Directive:

AuthDigestFile (→ p.111)

`mod_digest` avoids the transmission of passwords in clear text by using one-way message digests.

2.3.5 Content Selection

■ **mod_dir** (enabled by default)

Content Selection by Using Directory Default Documents

Since Apache 1.0, `src/modules/standard/mod_dir.c`

Rob McCool, Robert S. Thau (1993)

`mod_dir` performs a basic task of any web server: after some URL transformation has mapped a URL to a directory on the local file system, this module tries to select the default document inside this directory. It also solves a related problem: if the URL does not end in a slash (not `xxx/`) but was nevertheless mapped to a directory rather than a file, a slash is appended to the URL and an HTTP redirect is performed to avoid problems with relative hyperlinks inside the document.

Directive:

DirectoryIndex (→ p.111)

`mod_dir` solves the "trailing slash" problem.

■ **mod_actions** (enabled by default)

Content Selection by Content Types and Request Methods

Since Apache 1.1, `src/modules/standard/mod_actions.c`

Alexei Kosut (1996)

⁵<ftp://ftp.isi.edu/in-notes/rfc2617.txt>

`mod_actions` provides a way to trigger a CGI script when a specific MIME content type of a document is encountered or when the request uses a specific HTTP method. This module can be used to create dynamic content when specific documents are requested or to implement special extensional HTTP methods (such as PUT) via CGI scripts.

Directives:

Action (→ p.112) **Script** (→ p.112)

■ **mod_negotiation** (enabled by default)

Content Selection by Best-Matching Client Capabilities

Since Apache 1.0, `src/modules/standard/mod_negotiation.c`

Robert S. Thau, Roy T. Fielding (1995)

The HTTP protocol provides a flexible content negotiation facility controlled by the `Accept` and `Accept-XXX` headers. If `Options MultiViews` is active, the `mod_negotiation` module chooses the best representation of a resource (when a resource is available in several different representations, of course) based on the client-supplied preferences for media type, languages, character set, and encoding. It also implements features intended to provide more intelligent handling of requests for clients that send incomplete negotiation information. The internal algorithms in this module are very complex and partly even heuristic, which makes this module really nontrivial — both to understand and to use.

`mod_negotiation`
provides complex
content negotiations.

Directives:

CacheNegotiatedDocs (→ p.113) **LanguagePriority** (→ p.113)

2.3.6 Environment Creation

■ **mod_env** (enabled by default)

Fixed Environment Variable Creation

Since Apache 1.1, `src/modules/standard/mod_env.c`

Andrew Wilson (1995)

`mod_env` is a very simple module that performs one basic task: it controls the export of variables to the SSI/CGI environment and allows the webmaster to force values of variables.

`mod_env` controls the
CGI environment.

Directives:

PassEnv (→ p.114) **UnsetEnv** (→ p.114)
SetEnv (→ p.114)

■ **mod_setenvif** (enabled by default)

Conditional Environment Variable Creation

Since Apache 1.2, `src/modules/standard/mod_setenvif.c`

Alexei Kosut, Paul Sutton (1996)

`mod_setenvif` is a more advanced module for setting SSI/CGI environment variables. It sets variables depending on various client information found in the HTTP request. This ability is useful when combined with a special variable-based feature of the core module's `deny` and `allow` directives. In addition, some internal HTTP protocol behaviors of Apache can be controlled through variables that are usually set with the `SetEnvIf` directive of this module.

Directives:

BrowserMatch (→ p.114)

SetEnvIf (→ p.115)

BrowserMatchNoCase (→ p.115)

SetEnvIfNoCase (→ p.116)

`mod_setenvif`
conditionally sets
environment variables.

■ **mod_unique_id** (disabled by default)

Generation of Unique Identifiers by Request

Since Apache 1.3, `src/modules/standard/mod_unique_id.c`

Dean Gaudet, Alvaro Martinez Echevarria (1997)

`mod_unique_id` performs a simple but sometimes useful task: it generates a magic token for each HTTP request that is guaranteed to be unique across *all* requests under very specific conditions. The identifier will even be unique across multiple machines in a properly configured cluster of machines. It is exported to the SSI/CGI environment as a variable.

Directives: *none*

`mod_unique_id`
generates unique
identifiers for each
request.

2.3.7 Server-Side Scripting

■ **mod_cgi** (enabled by default)

Common Gateway Interface (CGI) Implementation

Since Apache 1.0, `src/modules/standard/mod_cgi.c`

Rob McCool, Robert S. Thau (1995)

The *Common Gateway Interface* (CGI) is the classical interface for generating dynamic content on the web. It is basically a set of environment variables that the server provides to a program while spawning a specific request to create the dynamic content. CGI remains the only truly portable and standardized scripting environment provided by web servers. Unfortunately, this way of creating dynamic content requires many resources (spawning a subprocess costs many extra memory and CPU cycles) and increases the response time. Alternative solutions are possible, for instance, with `mod_perl`.

`mod_cgi` provides the
Common Gateway
Interface.

Directives:

ScriptLog (→ p.116)

ScriptLogBuffer (→ p.117)

ScriptLogLength (→ p.117)

■ **mod_include** (enabled by default)

Server-Side Includes (SSI) Implementation

Since Apache 1.0, `src/modules/standard/mod_include.c`

The Apache Group (1995)

`mod_include` implements an extended version of the *Server-Side Includes* (SSI) quasi-standard, (which was originally introduced by the NCSA httpd). Embedded programming constructs in an HTML document are evaluated “on the fly” and expanded by the server before the document is sent to the client. The name of the module relates to a major goal of SSI — its role as a file inclusion facility — but please note that using this facility decreases server performance.

`mod_include` provides Server-Side Includes.

Directive:

XBitHack (→ p.117)

2.3.8 Response Header Generation

■ **mod_mime** (enabled by default)

Fixed Content Type/Encoding Assignment

Since Apache 1.0, `src/modules/standard/mod_mime.c`

Rob McCool (1995)

`mod_mime` provides for static assigning of MIME content types and content encodings to documents. This assignment is primarily carried out through the file extensions of the documents. For more advanced mappings, one can use `mod_mime_magic`. Nevertheless, `mod_mime` is very important in Apache, because many other modules depend on `mod_mime` content-type tables.

`mod_mime` assigns MIME types to documents.

Directives:

AddEncoding (→ p.118)

AddHandler (→ p.118)

AddLanguage (→ p.118)

AddType (→ p.119)

DefaultLanguage (→ p.119)

ForceType (→ p.119)

RemoveHandler (→ p.120)

SetHandler (→ p.120)

TypesConfig (→ p.120)

■ **mod_mime_magic** (disabled by default)

Automatic Content Type/Encoding Assignment

Since Apache 1.3, `src/modules/standard/mod_mime_magic.c`

Ian F. Darwin, Ian Kluff (1997)

`mod_mime_magic` is a nifty variant of `mod_mime` that guesses MIME types by inspecting the first bytes of a document.

`mod_mime_magic` provides for dynamic assignment of MIME content types and encodings to documents. In contrast to `mod_mime`, this module looks directly at the content of the document (usually its first bytes) and tries to guess the content type and encoding by matching so-called magic cookies (byte sequences that are unique to particular file formats). The resulting performance penalty should not be neglected. Thus, it is a good idea to manually define as many commonly known and straight forward MIME-types as possible with `mod_mime`, so `mod_mime_magic` has to determine only the remaining ones.

Directive:

MimeMagicFile (→ p.121)

■ `mod_expires` (disabled by default)

Creation of HTTP Expires Header

Since Apache 1.2, `src/modules/standard/mod_expires.c`

Andrew Wilson (1996)

`mod_expires` controls the setting of the HTTP Expires header field in server responses. The expiration date can be set relative to either the time of the source document's last modification, or the time of the last client access. This information informs the client and intermediate HTTP proxies about the document's validity and persistence. If cached, the document may be fetched from the cache rather than from the source until the expiration date has passed. After that time, the cache copy is considered "expired" and invalid, and a new copy must be obtained from the source.

Directives:

ExpiresActive (→ p.121) **ExpiresDefault** (→ p.122)

ExpiresByType (→ p.121)

■ `mod_headers` (disabled by default)

Creation of Arbitrary HTTP Headers

Since Apache 1.2, `src/modules/standard/mod_headers.c`

Paul Sutton (1996)

`mod_headers` can add arbitrary HTTP header fields in the server response. Any header field can be replaced, deleted, or extended. It therefore allows you to tailor arbitrary HTTP responses.

Directive:

Header (→ p.122)

■ `mod_cern_meta` (disabled by default)

Creation of Arbitrary HTTP Headers (CERN-style)

Since Apache 1.1, `src/modules/standard/mod_cern_meta.c`

Andrew Wilson (1996)

`mod_expires` provides and controls the HTTP Expires header field.

`mod_headers` allows you to set arbitrary HTTP response header fields.

`mod_cern_meta` is a backward-compatible module that provides approximately the same functionality as `mod_headers`, albeit in a somewhat different way. Instead of being configured through Apache directives, the headers are placed inside a dedicated *meta*-file.

`mod_cern_meta` is a variant of `mod_headers` that uses external files.

Directives:

MetaDir (→ p.123) **MetaSuffix** (→ p.123)
MetaFiles (→ p.123)

2.3.9 Internal Content Handlers

■ `mod_asis` (enabled by default)

Generation of Raw Responses

Since Apache 1.0, `src/modules/standard/mod_asis.c`

The Apache Group (1995)

`mod_asis` allows file types to be defined so that Apache will send them *as is*, without adding HTTP headers. It can be used to send any kind of data from the server, including redirects and other special HTTP responses, without the use of a CGI program.

`mod_asis` allows you to send out arbitrary HTTP responses.

Directives: *none*

■ `mod_autoindex` (enabled by default)

Generation of Directory Index Documents

Since Apache 1.0, `src/modules/standard/mod_autoindex.c`

Rob McCool, Robert S. Thau (1993)

The index document of a directory (requested with a URL ending in a slash) can come from one of two sources: a file written by the user or a listing generated by the server. This module uses the latter source. The layout of such on-the-fly generated index listings can be controlled in many different ways.

`mod_autoindex` is Apache's built-in "ls -l" function.

Directives:

AddAlt (→ p.124)	DefaultIcon (→ p.126)
AddAltByEncoding (→ p.124)	FancyIndexing (→ p.126)
AddAltByType (→ p.125)	HeaderName (→ p.127)
AddDescription (→ p.125)	IndexIgnore (→ p.127)
AddIcon (→ p.125)	IndexOptions (→ p.127)
AddIconByEncoding (→ p.126)	IndexOrderDefault (→ p.128)
AddIconByType (→ p.126)	ReadmeName (→ p.128)

■ `mod_status` (enabled by default)

Display Summary of Server Runtime Information

Since Apache 1.1, `src/modules/standard/mod_status.c`

Mark Cox (1995)

`mod_status` provides a content handler that can be mapped to a URL and that outputs a runtime status page when requested. In particular,

internal process information is displayed. A variant can be displayed that gives a simple machine-readable list of the current server state. Always think about your privacy before using this module.

Directive:

ExtendedStatus (→ p.129)

■ **mod_info** (disabled by default)

Display Summary of Server Configuration-Time Information

Since Apache 1.1, `src/modules/standard/mod_info.c`

Rasmus Lerdorf, Lou Langholtz (1996)

`mod_info` provides a content handler that can be mapped to a URL and that outputs a configuration-time information page when requested. This page includes the compiled-in modules and all configuration directives that are present in the server configuration files.

Directive:

AddModuleInfo (→ p.129)

`mod_info` shamelessly publishes your server configuration.

2.3.10 Request Logging

■ **mod_log_config** (enabled by default)

Generic Request Logging

Since Apache 1.0, `src/modules/standard/mod_log_config.c`

Robert S. Thau (1995)

`mod_log_config` provides for logging of the requests made to the web server, using the *Common Log Format* (CLF) or any other user-specified format. It uses `printf(3)`-style format strings to define the log file entries. These entries can be written to a file or a reliable pipe connected to a spawned program.

Directives:

CookieLog (→ p.129) **LogFormat** (→ p.130)
CustomLog (→ p.130) **TransferLog** (→ p.130)

`mod_log_config` allows the writing of custom log files.

■ **mod_log_agent** (disabled by default)

Specialized User-Agent Logging (Deprecated)

Since Apache 1.0, `src/modules/standard/mod_log_agent.c`

The Apache Group (1995)

`mod_log_agent` is deprecated because it has been superseded by `mod_log_config`. It provides logging of the *User-Agent* HTTP header information.

Directive:

AgentLog (→ p.131)

■ **mod_log_referer** (disabled by default)

Specialized Referrer Logging (Deprecated)

Since Apache 1.0, `src/modules/standard/mod_log_referer.c`

The Apache Group (1995)

`mod_log_referer` is deprecated because it has been superseded by `mod_log_config`. It provides logging of the `Referer` HTTP header information.

Directives:

RefererIgnore (→ p.131) **RefererLog** (→ p.131)

■ **mod_usertrack** (disabled by default)

Specialized User Click-Trail Logging

Since Apache 1.0, `src/modules/standard/mod_usertrack.c`

Mark J. Cox (1995)

`mod_usertrack` is a module that implements a nifty idea, but that suffers from nasty side effects in practice. It generates a *clickstream* log of user activity on the server by using HTTP *cookies* (information included by the server in responses that is stored by the client and sent back to the server on subsequent requests). Unfortunately, not all clients support cookies. In addition, many clients, by default, require an interactive user dialog to accept cookies. The use of HTTP cookies defeats caching, too. These facts make the theoretically useful facility mostly unusable in practice.

`mod_usertrack` tracks user activity via HTTP cookies.

Directives:

CookieExpires (→ p.132) **CookieTracking** (→ p.132)

CookieName (→ p.132)

2.3.11 Experimental

■ **mod_mmap_static** (disabled by default)

Caching of Frequently Served Pages via Memory Mapping

Since Apache 1.3, `src/modules/experimental/mod_mmap_static.c`

Dean Gaudet (1997)

Serving static pages from disk is the main task of a web server. The `mod_mmap_static` module maps a statically configured list of frequently requested (but not changed) documents into memory by using the UNIX `mmap(2)` function. Although this approach can dramatically reduce the response time and I/O consumption, it unfortunately brings nasty

`mod_mmap_static` is experimental and memory-maps documents for fastest serving.

problems: every time a file changes, a reload of the server is required to remap the new contents into memory because `mmap(2)` doesn't allow automatic refreshments.

Directive:

MMapFile (→ p.133)

■ **mod_example** (disabled by default)

Apache API Demonstration (Developers Only)

Since Apache 1.2, `src/modules/example/mod_example.c`

Ken Coar (1997)

`mod_example` is a demonstration-only module. It allows Apache module authors to easily explore the Apache Module API. The module basically hooks into every API processing phase, enabling developers to observe the API processing steps. It should never be used within a production server, of course.

`mod_example` lets developers learn the internal processing stages of the Apache API.

Directive:

Example (→ p.133)

2.3.12 Extensional Functionality

■ **mod_proxy** (disabled by default)

Caching Proxy Implementation for HTTP and FTP

Since Apache 1.1, `src/modules/proxy/mod_proxy.c`

Ben Laurie, Chuck Murcko (1996)

`mod_proxy` implements a caching proxy inside Apache. This proxy facility can be used either as a real web proxy by the clients or as a back end by other modules (such as `mod_rewrite`) for performing HTTP client tasks. Although this module opens the door to a lot of nifty solutions, it currently remains a “stepchild” inside Apache. It was designed when HTTP/1.0 was considered state of the art. With HTTP/1.1, however, the requirements for a proxy changed dramatically. Also, with the initial design one could not provide real HTTP/1.1-conforming proxy functionality. Thus, although Apache (as an origin server) is fully HTTP/1.1 compliant, the proxy module is just HTTP/1.0 compliant. Whereas this module is no longer used to establish a real proxy server, it remains of interest when applied in conjunction with other modules like `mod_rewrite` or `mod_ssl`.

`mod_proxy` is one of the stepchild modules of Apache.

Directives:

AllowCONNECT (→ p.134)	NoCache (→ p.137)
CacheDefaultExpire (→ p.138)	NoProxy (→ p.134)
CacheDirLength (→ p.138)	ProxyBlock (→ p.135)
CacheDirLevels (→ p.138)	ProxyDomain (→ p.135)
CacheForceCompletion (→ p.140)	ProxyPass (→ p.136)
CacheGclInterval (→ p.139)	ProxyPassReverse (→ p.137)
CacheLastModifiedFactor (→ p.139)	ProxyReceiveBufferSize (→ p.136)
CacheMaxExpire (→ p.139)	ProxyRemote (→ p.135)
CacheRoot (→ p.137)	ProxyRequests (→ p.134)
CacheSize (→ p.138)	ProxyVia (→ p.136)

■ **mod_perl** (disabled by default)

Perl Integration and Interface

Since Apache 1.1, `src/modules/perl/mod_perl.c` (third-party)

Doug MacEachern (1996)

The `mod_perl` third-party module integrates the Perl programming language into the Apache web server by providing the Apache Module API (written in ANSI C) at the Perl programming level. One can therefore easily extend the Apache web server by writing extensions in Perl, which is a much easier task than writing an Apache module in ANSI C. Many interesting Perl modules for Apache (usually named `Apache::XXX`) already exist and provide high-level features for Apache. One of the most interesting standard use cases for `mod_perl` is `Apache::Registry`: In an alternative CGI implementation, the CGI scripts (written in Perl) are precompiled into byte-code, cached, and on request executed inside the address space of the Apache process. With this approach, the response time is a magnitude faster and the resource requirements are reduced.

`mod_perl` combines the flexibility of Apache with the programming power of Perl.

Directives:

</Perl> (→ p.141)	PerlLogHandler (→ p.148)
<Perl> (→ p.141)	PerlModule (→ p.143)
=cut (→ p.141)	PerlOpmask (→ p.143)
=pod (→ p.141)	PerlPassEnv (→ p.144)
PerlAccessHandler (→ p.147)	PerlPostReadRequestHandler (→ p.146)
PerlAuthenHandler (→ p.147)	PerlRequire (→ p.143)
PerlAuthzHandler (→ p.147)	PerlRestartHandler (→ p.150)
PerlChildExitHandler (→ p.149)	PerlSendHeader (→ p.145)
PerlChildInitHandler (→ p.145)	PerlSetEnv (→ p.144)
PerlCleanupHandler (→ p.149)	PerlSetVar (→ p.144)
PerlDispatchHandler (→ p.149)	PerlSetupEnv (→ p.144)
PerlFixupHandler (→ p.148)	PerlTaintCheck (→ p.142)
PerlFreshRestart (→ p.142)	PerlTransHandler (→ p.146)
PerlHandler (→ p.148)	PerlTypeHandler (→ p.147)
PerlHeaderParserHandler (→ p.146)	PerlWarn (→ p.142)
PerlInitHandler (→ p.145)	__END__ (→ p.141)

■ **mod_ssl** (disabled by default)

SSL/TLS Integration and Interface

Since Apache 1.3, `src/modules/ssl/mod_ssl.c` (third-party)
Ralf S. Engelschall (1998)

The `mod_ssl` third-party module provides strong cryptography via the *Secure Sockets Layer* (SSL version 2/3) and *Transport Layer Security* (TLS version 1) protocols with the help of the SSL/TLS implementation toolkit OpenSSL. SSL/TLS is a generic cryptography protocol that resides on top of TCP/IP (but below protocols like HTTP) and is used inside web servers to provide the HTTPS protocol (which mainly is HTTP over SSL/TLS over TCP/IP). The `mod_ssl` module here forms the essential glue code between the OpenSSL toolkit and the Apache web server.

Directives:

SSLCACertificateFile (→ p.154)	SSLOptions (→ p.157)
SSLCACertificatePath (→ p.154)	SSLPassPhraseDialog (→ p.150)
SSLCARevocationFile (→ p.155)	SSLProtocol (→ p.152)
SSLCARevocationPath (→ p.154)	SSLRandomSeed (→ p.151)
SSLCertificateFile (→ p.153)	SSLRequire (→ p.158)
SSLCertificateKeyFile (→ p.153)	SSLRequireSSL (→ p.157)
SSLCipherSuite (→ p.152)	SSLSessionCache (→ p.151)
SSLEngine (→ p.152)	SSLSessionCacheTimeout (→ p.152)
SSLLog (→ p.156)	SSLVerifyClient (→ p.155)
SSLLogLevel (→ p.156)	SSLVerifyClientDepth (→ p.155)
SSLMutex (→ p.150)	

`mod_ssl` combines the flexibility of Apache with the security of OpenSSL.